Splunk® Enterprise Search Manual 7.1.1

Generated: 10/09/2018 10:50 am

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Search Overview

Get started with Search

This manual discusses the **Search & Reporting app** and how to use the Splunk search processing language (**SPL**).

The Search app, the short name for the Search & Reporting app, is the primary way you navigate the data in your Splunk deployment. The Search app consists of a web-based interface (Splunk Web), a command line interface (CLI), and the Splunk SPL.

Start Here

If you are new to Splunk Search, the best way to get acquainted is to start with the Search Tutorial. The Search Tutorial introduces you to the Search and Reporting app and guides you through adding data, searching your data, and building simple reports and dashboards.

The Search Tutorial provides a great foundation for understanding Splunk Search.

Getting started in your own environment

After you complete the Search Tutorial, you should learn about the types of data you can explore, how Splunk software indexes data, and about Splunk knowledge objects.

Here are the resources to look at:

- Upload data to your Splunk deployment. See the Getting Data In manual.
- Understand how indexing works. See the Managing Indexers and Clusters of Indexers manual.
- Understand fields and knowledge objects, such as host, source type, and event type. See the Knowledge Manager Manual.

Use the Search app effectively

And of course you need to learn how to use the Search app effectively, which is the focus of this manual. This manual contains detailed information about how to search your data.

Basic Search app skills

- Navigating Splunk Web
- Using the Search app
- Types of searches
- Types of commands

Detailed Search information

- Retrieving events
- Specifying time ranges
- Optimizing searches
- Creating tables and charts
- Evaluating and manipulating fields
- Calculating statistics and advanced statistics
- Grouping and correlating events
- Managing search jobs

Search command reference

For a catalog of search commands and arguments that make up the Splunk SPL, see the *Search Reference*.

Distributed Search

If you are using Splunk Enterprise, **distributed search** provides a way to scale your deployment by separating the search management and presentation layer from the indexing and search retrieval layer. For an introduction to distributed search, see the *Distributed Search Manual*.

See also

Navigating Splunk Web
Using Splunk Search

Navigating Splunk Web

This topic discusses navigating the different views in **Splunk Web**, the Splunk web browser interface.

About Splunk Home

Splunk Home is your interactive portal to the data and apps in your Splunk deployment. The first time you log into your Splunk deployment, you land in Splunk Home. All of your apps appear on this page.

The main parts of Splunk Home include the navigation bar, the Apps menu, the Explore panel, and a custom default dashboard (not shown here).

Your Splunk account might be configured to start in another view instead of Splunk Home, such as Search or Pivot in the **Search & Reporting** app.

Apps panel

The Apps panel lists the apps that are installed on your Splunk instance that you have permission to view. Select the app from the list to open it. The **Search & Reporting** app is often referred to as Splunk Search. When you have more than one app, you can drag and drop the apps within the workspace to rearrange them.

You can perform the following actions.

- Click the gear icon to view and manage the apps that are installed in your Splunk deployment.
- Click the plus icon to browse for more apps to install.

Explore panel

The options in the Explore panel help you to get started. Click on the icons to open the **Add Data** view, browse for new apps, open the user documentation, or open Splunk Answers.

Home dashboard

Below the Explore panel is the home dashboard. When you first open Splunk Home, there is no default dashboard.

Click in the area labeled **Choose a home dashboard** to select a default dashboard.

If you are new to Splunk software, hold off selecting a default dashboard until you have created and saved a few searches. You might want to create a dashboard of your own and use that as your default dashboard.

For more information about dashboards, see the *Dashboards and Visualizations* manual.

About the Splunk bar

Use the Splunk bar to navigate Splunk Web. You can use it return to switch between apps, add data, manage settings and edit your Splunk configuration, view system-level messages, monitor the activity of your search jobs and alerts, and get help using Splunk software.

The Splunk bar in another view, such as the Search view in the **Search & Reporting** app, also includes an **App** menu next to the Splunk logo. Use the **App** menu to quickly switch between the Splunk applications that you have installed on your computer.

Return to Splunk Home

Click the Splunk logo on the navigation bar to return to Splunk Home from any other view in Splunk Web.

Settings menu

The Settings menu lists the configuration pages for Knowledge objects, Distributed environment settings, System and licensing, Data, and Authentication settings. If you do not see some of these options, you do not have the permissions to view or edit them.

Account menu

Use the **Account** menu to edit your account settings or log out of this Splunk installation. The **Account menu** is called "Administrator" because that is the default user name for a new installation. You can change this display name by selecting **Edit account** and changing the **Full name**. Other settings you can edit include: the time zone settings, the default app for this account, and the account's password.

Messages menu

All system-level error messages are listed on the **Messages** menu. When you have a new message to review, a numerical notification appears next to the **Messages** menu. The notification indicates the number of messages that you have.

Activity menu

The Activity menu lists shortcuts to the Jobs and Triggered alerts views.

- Click **Jobs** to open the search jobs manager window, where you can view and manage currently running searches.
- Click **Triggered Alerts** to view scheduled alerts that are triggered.

Help

Click **Help** to see links to Video Tutorials, Splunk Answers, the Splunk Support Portal, and online Documentation.

Find

Use **Find** to search for objects within your Splunk deployment. **Find** performs matches that are not case sensitive on the ID, labels, and descriptions in saved objects. For example, if you type **error**, it returns the saved objects that contain that term.

These saved objects include **Reports** and **Dashboards**. The results appear in the list separated by the categories where they exist.

You can also run a search for **error** in the **Search & Reporting** app by clicking **Open error in search**.

See also

Using Splunk Search

About the search language

The Splunk **Search Processing Language** (SPL) encompasses all the search commands and their functions, arguments and clauses. Search commands tell Splunk software what to do to the events you retrieved from the indexes. For example, you need to use a command to filter unwanted information, extract more information, evaluate new fields, calculate statistics, reorder your results, or create a chart.

Some search commands have functions and arguments associated with them. Use these functions and their arguments to specify how the commands act on your results and which fields they act on. For example, you can use functions to format the data in a chart, describe what kind of statistics to calculate, and specify what fields to evaluate. Some commands also use clauses to specify how to group your search results.

To get familiar with SPL, read these topics in this manual:

- Types of searches
- Types of commands
- Using Splunk Search

For more details on SPL syntax, see Understanding SPL syntax, in the *Search Reference*.

For information about functions, see

- Evaluation functions in the *Search Reference*
- Statistical and charting functions in the *Search Reference*

Types of searches

As you search, you will begin to recognize patterns and identify more information that can be useful as searchable fields. You can configure Splunk software to recognize these new fields as you index new data, or you can create new fields as you search. Whatever you learn, you can use, add, and edit this knowledge about fields, events, and transactions to your event data. This capturing of knowledge helps you to construct more efficient searches and build more detailed reports.

Before delving into the language and syntax of search, you should ask what you are trying to accomplish. Generally, after getting data into your Splunk deployment, you want to:

- Investigate to learn more about the data you just indexed or to find the root cause of an issue.
- Summarize your search results into a report, whether tabular or other visualization format.

Because of this, you might hear us refer to two types of searches: Raw event searches and transforming searches.

Raw event searches

Raw event searches are searches that just retrieve events from an index or indexes, and are typically used when you want to analyze a problem. Some examples of these searches include: checking error codes, correlating events, investigating security issues, and analyzing failures. These searches do not usually include search commands (except `search`, itself), and the results are typically a list of raw events.

- Read more about raw event searches starting with the topic About retrieving events.

Transforming searches

Transforming searches are searches that perform some type of statistical calculation against a set of results. These are searches where you first retrieve events from an index and then pass the events into one or more search commands. These searches will always require fields and at least one of a set of statistical commands. Some examples include: getting a daily count of error

events, counting the number of times a specific user has logged in, or calculating the 95th percentile of field values.

- Read more about the structure of a search in About the search processing language syntax.
- Read more about using subsearches to filter results in About subsearches.
- Read more about transforming searches and commands starting with the topic About transforming commands and searches.

Information density

Whether you are retrieving raw events or building a report, you should also consider whether you are running a search for *sparse* or *dense* information:

- **Sparse searches** are searches that look for a single event or an event that occurs infrequently within a large set of data. You have probably heard these referred to as 'needle in a haystack' or "rare term" searches. Some examples of these searches include: searching for a specific and unique IP address or error code.
- **Dense searches** are searches that scan through and report on many events. Some examples of these searches include: counting the number of errors that occurred or finding all events from a specific host.

See How search types affect Splunk Enterprise performance in the *Capacity Planning Manual*.

Types of commands

As you learn about Splunk SPL, you might hear the terms streaming, generating, transforming, and orchestrating used to describe the types of search commands. This topic explains what these terms mean and lists the commands that fall into each category.

There are six broad categorizations for almost all of the search commands:

- distributable streaming
- centralized streaming
- transforming
- generating

- orchestrating
- dataset processing

These categorizations are not mutually exclusive. Some commands fit into only one categorization. The `stats` command is an example of a command that fits only into the transforming categorization. Other commands can fit into multiple categorizations. For example a command can be streaming and also generating.

For a complete list of commands that are in each type, see Command types in the *Search Reference*.

Streaming and non-streaming commands

A **streaming command** operates on each event as it is returned by a search. Essentially one event in and one (or no) event out.

For example, the `eval` command can create a new field, `full_name`, to contain the concatenation of the value in the `first_name` field, a space, and the value in the `last_name` field.

```
... | eval full_name = first_name." ".last_name
```

The `eval` command evaluates each event without considering the other events.

A **non-streaming command** requires the events from all of the indexers before the command can operate on the entire set of events. Many transforming commands are non-streaming commands. There are also several commands that are not transforming commands but that are non-streaming. These non-transforming, non-streaming commands are sometimes referred to as event based non-streaming commands.

For example, before the `sort` command can begin to sort the events, the entire set of events must be received by the `sort` command. Other examples of non-streaming commands include `dedup`, `stats`, and `top`.

Non-streaming commands force the entire set of events to the search head. This requires a lot of data movement and a loss of parallelism.

For information on how to mitigate the cost of non-streaming commands, see Write better searches in this manual.

Processing attributes

The following table describes the processing differences between some of the types of commands.

	Distributable streaming	Centralized streaming	Data processing (non-streaming)	Transforming
Can run on indexers	Y	N	N	N
Can output before final input	Y	Y	N	N
Outputs events if inputs are events	Y	Y	Y	N

When a command is run it outputs either events or results, based on the type of command. For example, when you run the `sort` command, the input is events and the output is events in the sort order you specify. However, transforming commands do not output events. Transforming commands output results. For example the `stats` command outputs a table of calculated results. The events used to calculate those results are no longer available. After you run a transforming command, you can't run a command that expects events as an input.

Data processing commands are non-streaming commands that require the entire dataset before the command can run. These commands are not transforming, not distributable, not streaming, and not orchestrating. The `stats` command is an example of a data processing command. See Data processing commands.

Distributable streaming

A streaming command operates on each event returned by a search. For distributable streaming, the order of the events does not matter. A distributable streaming command is a command that can be run on the indexer, which improves processing time. The other commands in a search determine if the distributable streaming command is run on the indexer:

- If all of the commands before the distributable streaming command can be run on the indexer, the distributable streaming command is run on the indexer.
- If any one of the commands before the distributable streaming command must be run on the search head, the remaining commands in the search **must be** run on the search head. When the search processing moves to the search head, it cannot move back to the indexer.

Distributable streaming commands can be applied to subsets of indexed data in a parallel manner. For example, the `rex` command is streaming. It extracts fields and adds them to events at search time.

Some of the common distributable streaming commands are: eval, fields, makemv, rename, regex, replace, strcat, and where.

For a complete list of distributable streaming commands, see Streaming commands in the *Search Reference*.

Centralized streaming

For centralized streaming commands, the order of the events matters. A centralized streaming command applies a transformation to each event returned by a search. But unlike distributable streaming commands, a centralized streaming command only works on the search head. You might also hear the term "stateful streaming" to describe these commands.

Centralized streaming commands include: head, streamstats, some modes of dedup, and some modes of cluster.

Transforming

A **transforming command** orders the search results into a data table. These commands "transform" the specified cell values for each event into numerical values that Splunk software can use for statistical purposes. Transforming commands are not streaming. Also, transforming commands are required to transform search result data into the data structures that are required for visualizations such as column, bar, line, area, and pie charts.

Transforming commands include: chart, timechart, stats, top, rare, contingency, highlight, typer, and addtotals when it is used to calculate column totals (not row totals).

For more information about transforming commands and their role in create statistical tables and chart visualizations, see About transforming commands and searches in the this manual.

For a complete list of transforming commands, see Transforming commands in the *Search Reference*.

Generating

A **generating command** fetches information from the indexes, without any transformations. Generating commands are either event-generating (distributable or centralized) or report-generating. Most report-generating commands are also centralized. Depending on which type the command is, the results are returned in a list or a table.

Generating commands do not expect or require an input. Generating commands are usually invoked at the beginning of the search and with a leading pipe. That is, there cannot be a search piped into a generating command. The exception to this is the `search` command, because it is implicit at the start of a search and does not need to be invoked.

Examples of generating commands include: dbinspect, datamodel, inputcsv, metadata, pivot, search, and tstats

For a complete list of generating commands, see Generating commands in the *Search Reference*.

Orchestrating

An **orchestrating command** is a command that controls some aspect of how the search is processed. It does not directly affect the final result set of the search. For example, you might apply an orchestrating command to a search to enable or disable a search optimization that helps the overall search complete faster.

Examples of orchestrating commands include redistribute, noop, and localop. The lookup command also becomes an orchestrating command when you use it with the `local=t` argument.

Dataset processing

There are a handful of commands that require the entire dataset before the command can run. These commands are referred to as dataset processing

commands. These commands are not transforming, not distributable, not streaming, and not orchestrating. Some of these commands fit into other types in specific situations or when specific arguments are used.

Examples of data processing commands include: sort, eventstats, and some modes of cluster, dedup, and fillnull.

For a complete list of dataset processing commands, see Dataset processing commands in the *Search Reference*.

Search with Splunk Web, CLI, or REST API

You can perform searches using Splunk Web and the Splunk REST API. If you use Splunk Enterprise, you can also run a search from the command line interface (CLI). Which tool is best can sometimes depend on what you want from your search.

If you need to be able to search a Splunk Enterprise and Splunk Cloud deployment together in a single search, you must configure hybrid searching. See Configure hybrid search in the *Splunk Cloud User Manual*.

Search with Splunk Web

When you search with Splunk Web, you are using the Search app, and you can control the search experience by selecting a search mode (Fast, Verbose, Smart). Depending on the mode you select, Splunk software automatically discovers and extracts fields other than the default fields, returns results as an events list or a table, and runs the calculations required to generate the event timeline. Calculating the event timeline is very expensive because it creates buckets and keeps the statistics for events and fields in a dispatch directory such that it is available when the user clicks a bar on the timeline.

- Read more about how to "Set search mode to adjust your search experience" in this manual.

Search with the CLI or REST API

When you run a search through the command line interface (CLI) or use the search jobs endpoint in the REST API to create a search, the search goes directly to the Splunk search engine without going through **Splunk Web**. These searches can complete much faster than the searches in Splunk Web because

Splunk software does not calculate or generate the event timeline. Instead, search results are displayed as a raw events list or a table, depending on the type of search.

- Read more "About CLI searches" in the *Search Reference*.
- Read about "Creating searches using the REST API" in the *REST API Reference Manual*.

Using the Search App

About the Search app

The Search & Reporting app, referred to as the *Search app*, is the application that you use to search and create reports on your data.

This topic describes the views and elements that comprise the Search app.

Open the Search app

1. From Splunk Home, click **Search & Reporting** in the **Apps** panel. This opens the Search Summary view in the Search & Reporting app.

The Search summary view

Before you run a search, the Search summary view displays the following elements: App bar, Search bar, Time range picker, **How to Search** panel, **What to Search** panel, and the **Search History**.

Some of these are common elements that you see on other views. Elements that are unique to the Search Summary view are the panels below the Search bar: the **How to Search** panel, the **What to Search** panel, and the **Search History** panel.

Number	Element	Description
1		

	Applications menu	Switch between Splunk applications that you have installed. The current application, Search & Reporting app, is listed. This menu is on the Splunk bar.
2	**Splunk bar**	Edit your Splunk configuration, view system-level messages, and get help on using the product.
3	**Apps bar**	Navigate between the different views in the application you are in. For the Search & Reporting app the views are: Search, Datasets, Reports, Alerts, and Dashboards.
4	**Search bar**	Specify your search criteria.
5	**Time range picker**	Specify the time period for the search, such as the last 30 minutes or yesterday. The default is **Last 24 hours**.
6	**How to search**	Contains links to the *Search Manual' and the*Search Tutorial*.*
7	**What to search**	Shows a summary of the data that is uploaded on to this Splunk instance and that you are authorized to view.
8	**Search history**	View a list of the searches that you have run. The search history appears after you run your first search.

Data summary

The Data Summary dialog box shows three tabs: Hosts, Sources, Sourcetypes. These tabs represent searchable fields in your data.

Host

The **host** of an event is the host name, IP address, or fully qualified domain name of the network machine from which the event originated. In a distributed environment, you can use the host field to search data from specific machines.

Source

The **source** of an event is the file or directory path, network port, or script from which the event originated.

Source type

The **source type** of an event tells you what kind of data it is, usually based on how the data is formatted. This classification lets you search for the same type of data across multiple sources and hosts.

In this example, source types are:

- **access_combined_wcookie:** Apache web server logs
- **secure:** Secure server logs
- **vendor_sales:** Global sales vendors

For information about which source type is assigned to your data, see Why source types matter in the *Getting Data In* manual.

The New Search view

The **New Search** view opens after you run a search or when you click the **Search** tab to start a new search. The App bar, Search bar, and Time range picker are still available in this view. Additionally, this view contains many more elements: search action buttons and search mode selector; counts of events; job status bar; and tabs for Events, Patterns, Statistics, and Visualizations.

You can type `index=_internal` in the Search bar and press **Enter** to look at the events from the internal log files on your Splunk instance.

If you followed the steps to get data into your Splunk deployment in the *Search Tutorial*, you can type `buttercupgames` in the Search bar and press **Enter** to search for the "buttercupgames" keyword in your events.

In this view, the App bar, Search bar and Time range picker are also available. The **New Search** view contains many more elements such as search action buttons, a search mode selector, counts of events, a job status bar, and results tabs for Events, Patterns, Statistics, and Visualizations.

App bar

Use the App bar to navigate between the different views in the Search & Reporting app: Search, Pivot, Reports, Alerts, and Dashboards. There are entire manuals devoted to these other capabilities.

- *Alerting manual*
- *Dashboards and Visualizations*
- *Pivot manual*
- *Reporting manual*

Search bar

Use the search bar to specify your search criteria in Splunk Web. Type your search string and press **Enter**, or click the Search icon which is on the right side of the search bar.

Time range picker

Time is the single most important search parameter that you specify.

Use the time range picker to retrieve events over a specific time period. For **real-time searches** you can specify a window over which to retrieve events. For **historical searches**, you can restrict your search by specifying a relative time range such as 15 minutes ago, Yesterday, and so on. You can also restrict your searches using a specific date and time range. The time range picker has many preset time ranges that you can select from, but you can also type a custom time range.

For more information, see About searching with time.

Timeline

The timeline is a visual representation of the number of events that occur at each point in time in your results. Peaks or valleys in the timeline can indicate spikes in activity or server downtime. The timeline options are located above the timeline. You can zoom in, zoom out, and change the scale of the chart.

When you click a point on the timeline or use on of the timeline options, the display of the timeline changes based on the events returned from your search. A new search is not run.

Search actions

There are a wide range of search actions you can perform, including working with your search Jobs, saving, sharing, exporting, and printing your search results.

For more information, see:

- Perform actions on running searches
- About jobs and job management
- Export search results

Search mode

You can use the search mode selector to provide a search experience that fits your needs. The modes are Smart (default), Fast, and Verbose.

For more information, see Search modes.

Fields sidebar

To the left of the events list is the Fields sidebar. As events are retrieved that match your search, the Fields sidebar shows the **Selected Fields** and **Interesting Fields** in the events. These are the fields that the Splunk software extracts from your data.

When you first run a search the **Selected Fields** list contains the default fields host, source, and sourcetype. The default fields appear in every event.

Interesting Fields are fields that appear in at least 20% of the events.

Next to the field name is a count of the how many events the field name appears in. Click on any field name to show more information about that field.

Anatomy of a search

A search consists of a series of commands that are delimited by pipe (|) characters. The first whitespace-delimited string after each pipe character controls the command used. The remainder of the text for each command is handled in a manner specific to the given command.

This topic discusses an anatomy of a Splunk search and some of the syntax rules shared by each of the commands and syntax rules for fields and field

values.

The anatomy of a search

To better understand how search commands act on your data, it helps to visualize all your indexed data as a table. Each search command redefines the shape of your table.

For example, let's take a look at the following search.

```
sourcetype=syslog ERROR | top user | fields - percent
```

The Disk represents all of your indexed data. The Disk is a table of a certain size with columns that represent fields and rows that represent events. The first intermediate results table shows fewer rows--representing the subset of events retrieved from the index that matched the search terms "sourcetype=syslog ERROR". The second intermediate results table shows fewer columns, representing the results of the top command, "top user", which summarizes the events into a list of the top 10 users and displays the user, count, and percentage. Then, "fields - percent" removes the column that shows the percentage, so you are left with a smaller final results table.

About the search pipeline

The "search pipeline" refers to the structure of a Splunk search, in which consecutive commands are chained together using a pipe character, "|". The pipe character tells Splunk software to use the output or result of one command (to the left of the pipe) as the input for the next command (to the right of the pipe). This enables you to refine or enhance the data at each step along the pipeline until you get the results that you want.

A Splunk search starts with search terms at the beginning of the pipeline. These search terms are keywords, phrases, boolean expressions, key/value pairs, etc. that specify which events you want to retrieve from the index(es). See "About retrieving events".

The retrieved events can then be passed as inputs into a search command using a pipe character. Search commands tell Splunk software what to do to the events after you retrieved them from the index(es). For example, you might use commands to filter unwanted information, extract more information, evaluate new fields, calculate statistics, reorder your results, or create a chart. Some commands have functions and arguments associated with them. These functions and their arguments enable you to specify how the commands act on your results and which fields to act on; for example, how to create a chart, what kind of statistics to calculate, and what fields to evaluate. Some commands also enable you to use clauses to specify how you want to group your search results.

- For more information about what you can do with search commands, see "About the search processing language".
- In the *Search Reference*, For a list of search commands, see the "Command quick reference" and the individual search command reference topics for syntax and usage information.

Quotes and escaping characters

Generally, you need quotes around phrases and field values that include white spaces, commas, pipes, quotes, or brackets. Quotes must be balanced, an opening quote must be followed by an unescaped closing quote. For example:

- A search such as `error | stats count` will find the number of events containing the string error.
- A search such as `... | search "error | stats count"` would return the raw events containing the literal string error, a pipe, stats, and count, in that order.

Additionally, you want to use quotes around keywords and phrases if you don't want to search for their default meaning, such as Boolean operators and field/value pairs. For example:

- A search for the keyword AND without meaning the Boolean operator: `error "AND"`
- A search for this field/value phrase: `error "startswith=foo"`

The backslash character (\) is used to escape quotes, pipes, and itself. Backslash escape sequences are still expanded inside quotes. For example:

- The sequence \| as part of a search will send a pipe character to the command, instead of having the pipe split between commands.

- The sequence \" will send a literal quote to the command, for example for searching for a literal quotation mark or inserting a literal quotation mark into a field using rex.
- The \\ sequence will be available as a literal backslash in the command.

If Splunk software does not recognize a backslash sequence, it will not alter it.

- For example \s in a search string will be available as \s to the command, because \s is not a known escape sequence.
- However, in the search string \\s will be available as \s to the command, because \\ is a known escape sequence that is converted to \.

Asterisks, *, cannot be searched for using a backslash to escape the character. Splunk software treats the asterisk character as a major breaker. Because of this, it will never be in the index. If you want to search for the asterisk character, you will need to run a post-filtering regex search on your data:

```
index=_internal | regex ".*\*.*"
```

For more information about major breakers, read "Overview of event processing" in the Getting Data in Manual.

Examples

Example 1: The `myfield` field is created with the value of `6`.

```
... | eval myfield="6"
```

Example 2: The `myfield` field is created with the value of `"`.

```
... | eval myfield="\""
```

Example 3: The `myfield` field is created with the value of `\`.

```
... | eval myfield="\\"
```

Example 4: This search would produce an error because of unbalanced quotation marks.

```
... | eval myfield="\"
```

Fields

Events and results flowing through the Splunk search pipeline exist as a collection of fields. Fields can fundamentally come from the Splunk index -- _time as the time of the event, source as the filename, etc -- or can be derived from a wide variety of sources at search time -- eventtypes, tags, regex extractions using the `rex` command, totals coming from the `stats` command, etc.

For a given event, a given field name might be present or absent. If present, it might contain a single value or multiple values. Each value is a text string. Values might be of positive length (a string, or text) or zero length (empty strings, or "").

Numbers, for example, are strings that contain the number. For example, a field containing a value of the number 10 contains the characters 1 and 0: "10". Commands that take numbers from values automatically convert them internally to numbers for calculations.

Null field
: A null field is not present on a particular result or event. Other events or results in the same search might have values for this field. For example, the `fillnull` command adds a field and default value to events or results that lack fields present on other events or results in the search.

Empty field
: An empty field is shorthand for a field that contains a single value that is the empty string.

Empty value
: A value that is the empty string, or "". You can also describe this as a zero-length string.

Multivalue field
: A field that has more than one value. All non-null fields contain an ordered list of strings. The common case is that this is a list of one value. When the list contains more than one entry, it is a multivalue field. See "Manipulate and evaluate fields with multiple values" in the *Search Manual*.

Help building searches

The Splunk Search Processing Language (SPL) includes commands and functions that you can use to build searches. All of the commands and functions are documented in the *Search Reference*.

Splunk Web has several built-in features to help you build and parse searches.

- Search assistant modes
- Syntax highlighting
- Auto-format search syntax
- Numbering search lines
- Shortcuts

This topic discusses using the search assistant. See Help reading searches for information about syntax highlighting, auto-formatting, line numbers, and shortcuts.

Use the search assistant to build searches

When you type a few letters or a term into the Search bar, the search assistant shows you terms and searches that match what you typed.

The **Matching Terms** are based on the terms that are indexed from your data. The **Matching Searches** are based on your recent searches.

The list continues to update as you type.

To add an item in the list to your search criteria you can click on an item, or use the arrow keys to highlight the item and press **Enter**.

Search assistant modes

The search assistant has three modes: Full, Compact, and None. The default mode is Compact.

Compact mode

The Compact mode displays a list of matching terms and searches when you type. When you type a pipe (|) character to indicate that you want to use a command, a list of the SPL commands appears. You can type a letter to jump to the section of the list that begins with that letter. For example, if you type the letter **s**, the list displays all of the commands that begin with the letter **s**.

When you type a command, a list appears showing Command History and Matching Searches. Initially, the Command History shows some command examples. As you use a command in your searches, the Command History displays your uses of the command instead of the examples.

Below the list are a brief description of the command and an example. The **Learn More** link opens the *Search Reference* in a new window and displays documentation about the command.

To access the **Learn More** link from your keyboard, use your arrow keys to highlight the command or attribute name. Press **Tab** to highlight the **Learn More** link and then press **Enter** to activate the link.

If you type something after the command, the search assistant shows any command arguments or history that match what you type.

The search assistant can also show you the data type that an argument requires. Type the argument in the Search bar. Include the equal (=) symbol, if that is part of the argument syntax. In the following example, the search assistant shows that a <string> value is required for the `countfield` argument.

Full mode

The Full mode displays a list of matching terms and searches when you type, along with a count of how many times a term appears in your indexed data. This count tells you how many search results will be returned if you search on that term. If a term or phrase is not in the list, the term is not in your indexed data.

The Full mode also provides suggestions in the How To Search section on ways that you can retrieve events and use the search commands.

When you type a command in the Search bar, the list of matching terms and searches is replaced with the **Command History** list.

To add an item in the Command History list to your search criteria click on an item, or use the arrow keys to highlight the item and press **Enter**.

The search assistant displays a brief description of the command and several examples. There are two links next to the command description: Help and More.

- The **Help** link opens the *Search Reference* in a new window and displays documentation about the command.
- The **More** link expands the information about the command that is displayed on the screen.

When you select the **More** link, several new sections appear. The **Details** section provides a detailed description of the command. The **Syntax** section shows the basic syntax for the command. The **Related** section lists commands that are related to the command that you typed. If the command has complex syntax, click the **More** link next to the syntax to expand the syntax.

If you type something after the command, the search assistant shows any command arguments or history that match what you type.

The search assistant can show you the data type that an argument requires. Type the argument in the Search bar. Include the equal (=) symbol if that is part of the argument syntax. In the following example, the search assistant shows that

a <string> value is required for the `countfield` argument.

None mode

You can turn off the search assistant by changing the mode to **None**.

Change the search assistant mode

The default search assistant mode is Compact. You can change the search assistant mode or temporarily hide the search assistant while you build your search.

When you change the search assistant mode, the change affects only your user account.

Prerequisites

- If the Search bar contains a search that you have not run, run the search before you change the search assistant mode. Otherwise, the search is lost when you change modes. Running the search adds the search to the search history, where you can access it after you change the mode.
- If you have a Splunk Free license, you cannot change the search assistant mode. The User account menu, where the Preferences options resides, is not available in Splunk Free. To learn about what is and is not included in Splunk Free, see About Splunk Free in the *Admin manual*.

Steps

1. On the Splunk bar, select **[*User_account_name*] > Preferences**.
2. Click **SPL Editor**.
3. Verify that the **Advanced editor** is turned on.
4. For Search assistant, click on the mode that you want to use, **Full**, **Compact**, or **None**.

5. Click **Apply**.

Hide and display the search assistant

By default, the search assistant opens when you type something into the Search bar. You can turn off or hide the search assistant.

Turn off the search assistant

To turn off the search assistant, change the search assistant mode to **None**.

Hide the search assistant

The options for hiding the search assistant depend on the mode that you are using.

Compact mode
: You cannot hide the search assistant. You can only turn off the search assistant.

Full mode
: To hide the search assistant in Full mode, you turn off the **Auto Open** feature and collapse the search assistant drop-down.

1. In the search assistant window, click **Auto Open**. This removes the check mark next to **Auto Open**.
2. Click the collapse and expand button on the right side of the Search bar to hide the search assistant.

   The search assistant remains hidden until you use the expand button to show the search assistant again. See Unhide the search assistant window in this topic.

When you uncheck **Auto Open** and click the collapse button the search assistant is hidden, even when you start a new search or close and reopen Splunk Web. The search assistant remains hidden until you unhide it.

Unhide the search assistant

If the search assistant is hidden, click the expand button on the right side of the Search bar and click **Auto Open**.

If these steps do not unhide the search assistant window, then either the search assistant is turned off or there is no assistance for what you have typed into the Search bar.

To turn the Search Assistant back on, you need to change the search assistant mode to **Compact** or **Full**.

Change the default search assistant mode for all users

Individual users can change the default search assistant setting for themselves. The default search assistant mode can also be changed globally, for all users.

Prerequisites

- Only users with file system access, such as system administrators, can change the default search assistant mode for all users.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

Steps

1. Open the local `user-prefs.conf.spec.in` file for the Search app. For example `$SPLUNK_HOME/etc/apps/<app_name>/local`.

2. Under the **[general]** stanza, change the search assistant mode by selecting one of the other mode values. Choose from **full**, **compact**, or **none**. For example: `search_assistant=full`.
3. Restart the Splunk instance.

Help reading searches

Search strings can be long and difficult to read. The Search bar contains features to help you read, parse, or interpret the Splunk Search Processing Language (SPL) syntax. The syntax highlighting feature displays parts of SPL in different colors. Syntax highlighting is available in two different color themes.

In addition to color themes, you can use auto-formatting and line numbers to help read searches. There are keyboard shortcuts available to help you find information in your search syntax.

Syntax highlighting

With syntax highlighting the SPL commands, arguments, functions, and keywords are color-coded to make it easier to read a search.

Consider the following search.

```
sourcetype=access_* | timechart count(eval(action=purchase)) BY
productName usenull=false useother=false
```

With syntax highlighting turned on, searches can be easier to read. Syntax highlighting shows commands, arguments, functions, and keywords in different colors. The following image shows a search string with syntax highlighting.

Syntax validation

If a command, argument, function, or boolean operator is not spelled or capitalized correctly, the term is not highlighted in color. The lack of color alerts you to incorrect syntax.

If you specify an incorrect data type for an argument, the data type appears in red. For example, the `limit` argument for the `top` command expects an integer. If

you type `...|top limit=false` the term `false` is highlighted in red because it is not an integer.

Turn off syntax highlighting

You can turn syntax highlighting colors off by changing the color theme to **Black on White**. This is useful for people who have difficulty distinguishing between different colors.

You cannot turn off or change syntax highlighting if you have a Splunk Free license. See About Splunk Free in the *Admin manual.*

1. On the Splunk bar, select **[*User_account_name*] > Preferences**.
2. Click **SPL Editor**.
3. On the **Themes** tab, click **Black on White**.
4. Click **Apply**.

Color themes

You can change the appearance of your search criteria by specifying a color theme. There are several themes to choose from.

Theme name	Description	Notes

Black on White	White background. Black text. No other colors.	Useful for people who have difficulty distinguishing between different colors.
Light Theme	White background. Black text. Colors for commands, arguments, functions, keyword modifiers, and Boolean operators.	Default theme
Dark Theme	Black background. Light grey text. Colors for commands, arguments, functions, keyword modifiers, and Boolean operators.	

Color codes

The color coding that is used for the search syntax depends on the color theme that is implemented. The **Light** theme is the default theme. The color codes for the **Light** and **Dark** themes are described in the following table.

Syntax component	Color	Example
Commands	Blue	...timechart
Command arguments	Green	...timechart usenull=false
Functions	Pink	...timechart count
Keyword modifiers and Boolean operators	Orange	...timechart count BY productName

The following image shows syntax highlighting with the **Dark** theme.

Change the color theme

You change the color theme in the Search bar by using the account menu. You cannot change the color theme if you have a Splunk Free license. See About Splunk Free in the *Admin manual*.

1. On the Splunk bar, select **[*User_account_name*] > Preferences**.
2. Click **SPL Editor**.
3. On the **Themes** tab, select the color theme that you want to use.
4. Click **Apply**.

Auto-format search syntax

As you build a search, you can set up the Splunk software to format the search syntax as you type. Auto-format makes your searches more readable. Each pipe section is parsed onto a separate line. Any subsearches are indented.

The following image shows how a search appears in the Search bar when auto-format is turned off.

When auto-format is turned on, this same search is parsed as shown in the following image.

Characters that trigger auto-format

Character	Automatic formatting
Pipe (|)	The pipe is placed on a new line to separate each new piped section of your search criteria.
Left square bracket ([ )	The left square bracket, which signifies the start of a subsearch, is placed on a new line and indented several spaces.

If the pipe or left bracket is inside a quoted string, the auto-format is not triggered.

Turn on Search auto-format

By default, automatic formatting of search syntax is turned off. You can turn on the automatic formatting of the search syntax in the Settings dialog box.

Changing the options in the Settings dialog box changes the setting only for you. It does not impact the setting for other users.

1. On the Splunk bar, select **[*User_account_name*] > Preferences**.
2. Click **SPL Editor**.
3. On the **General** tab click **Search auto-format**.

4. Click **Apply**.

Auto-formatting is applied to new searches that you type into the Search bar. If you already have a search in the Search bar, use the Search bar shortcuts to apply auto-formatting to that search.

Why are my searches not auto-formatting?

The auto-format feature works on searches that you type into the Search bar. If you paste a search into the Search bar or select a search from **Search History**, the search is not automatically formatted even when the auto-format feature is turned on.

To apply auto-formatting to a search that you paste into the Search bar or select from Search History, use the following keyboard shortcut to apply auto-formatting to that search.

- On Linux or Windows use **Ctrl +** \
- On Mac OSX use **Command +** \

Number search lines

To make reading your searches easier, you can display line numbers in the Search bar. The following image shows both line numbers and auto-formatting turned on.

Turn on line numbering

By default, line numbering is turned off. You turn on line numbering in the Preferences dialog box.

1. On the Splunk bar, select **[*User_account_name*] > Preferences**.
2. Click **SPL Editor**.
3. On the **General** tab click **Line numbers**.
4. Click **Apply**.

Changing the options in the **Preferences** dialog box changes the setting only for you. It does not impact the preferences set for other users. See Change the default Search preferences for all users.

A row in the Search bar is not a line

The line numbering feature applies numbers only to lines. A row in the Search bar is not necessarily a line. You might have a long line that spans multiple rows in the Search bar but is still only one line.

For example, if you paste a long search into the Search bar that has not been formatted with multiple lines, the search has one line number and spans multiple rows.

You can create lines in the Search bar by using the following methods.

- The Search auto-formatting feature is turned on and you type a pipe character or left square bracket.
- You use the Search bar shortcuts to auto-format the current search.
- You press **Shift + Enter** to split the active row at the cursor. Pressing **Enter** does not create a new line in the Search bar.

Search bar shortcuts

In the Search bar, you can use keyboard shortcuts to help you develop, read, and parse your search criteria.

Make searches easier to read

Long searches can be difficult to read. For example, the following search uses multiple commands and includes many occurrences of renaming columns in the search results.

```
sourcetype=access_* status=200 | stats count AS views
count(eval(action="addtocart")) AS addtocart
count(eval(action="purchase")) AS purchases by productName | eval
viewsToPurchases=(purchases/views)*100 | eval
cartToPurchases=(purchases/addtocart)*100 | table productName views
addtocart purchases viewsToPurchases cartToPurchases | rename
productName AS "Product Name", views AS "Views", addtocart as "Adds To
Cart", purchases AS "Purchases"
```

The following image shows how this search appears in the Search bar.

You can use a keyboard shortcut to parse each pipe section on a separate line. Any subsearches are indented. The auto-format feature does not need to be turned on to use these keyboard shortcuts.

- On Linux or Windows use **Ctrl +** \
- On Mac OSX use **Command +** \

The results of the shortcut are shown in the following image.

You can also use **Shift + Enter** to force a new line. See Line and word shortcuts.

Expand your search

For long searches, or searches that contain **search macros** or **saved searches**, it can be difficult to see the entire search in the Search bar.

You can see the contents of your entire search by using a keyboard shortcut, **Command-Shift-E (Mac OSX)** or **Control-Shift-E (Linux or Windows)** from the Search bar in the Search page. This opens a preview that displays the expanded search string, including all search macros and saved searches. If syntax highlighting or line numbering are turned on, those features also appear in the preview.

You can copy parts of the search from the preview window. You can also click **Open in Search** in the preview window to run your search in a new window. See Preview your search.

Highlight search terms

- To highlight all of the occurrences of a word in the search, double-click on that word.

Locate matching parenthesis

- Position your cursor immediately after an open or close parenthesis. The matching parenthesis is highlighted.

Undo and Redo shortcuts

Use these keyboard shortcuts to undo and redo actions in the Search bar.

Action	Linux or Windows	Mac OSX
Undo the previous action.	Ctrl + Z	Command + Z
Redo the previous action.	Ctrl + Y or Ctrl + Shift + Z	Command + Y or Command + Shift + Z

Search assistant window shortcuts

With the Compact mode of the search assistant, you can use keyboard shortcuts to select items in the list and close and reopen the search assistant window.

Action	Linux or Windows	Mac OSX
Move your cursor into the search assistant window.	Down arrow key	Down arrow key
Close the search assistant window.	ESC	ESC
Reopen the search assistant window.	Ctrl + Space	Control + Space
Select an item in the search assistant window and insert it into the Search bar.	Use the Up arrow and Down arrow keys to highlight the item and press **Enter**.	Use the Up arrow and Down arrow keys to highlight the item and press **Enter**.
Toggle between the list and the **Learn More** link in the search assistant window.	Tab	Tab

Find and replace shortcuts

Use the following keyboard shortcuts to find and replace terms in the Search bar.

Action	Linux or Windows	Mac OSX
Find a term.	Ctrl + F	Command + F
Find and replace a term.	Ctrl + H	Command + Option + F

Line and word shortcuts

The distinction between rows and lines is important to understand when you use keyboard shortcuts to manipulate rows or lines in your search criteria in the Search bar.

- Long searches appear on multiple rows in the Search bar.
- If the search is not parsed, the search is one line.
- If the search is parsed, separating each piped section and subsearch into its own line, a row is the same as a line.

Action	Linux or Windows	Mac OSX
Split the active row at the cursor.	Shift + Enter	Shift + Enter
Remove the active line. If the search is one line with multiple rows and not parsed into separate lines, the entire search is removed.	Ctrl + D	Command + D
Copy the active row and place the copy below the active row.	Alt + Shift + Down arrow	Command + Option + Down arrow
Copy the active row and place the copy above the active row.	Alt + Shift + Up arrow	Command + Option + Up arrow
Move the active row down one row.	Alt + Down arrow	Option + Down arrow
Move the active row up one row.	Alt + Up arrow	Option + Up arrow
Remove the search criteria from the cursor to the end of the row.	Alt + Delete	Control + K
Remove the search criteria from the cursor to the start of the row.	Alt + Backspace	Command + Delete
Remove the word or space to the right of the cursor.	Ctrl + Delete	Alt + Delete
Remove the word or space to the left of the cursor.	Ctrl + Backspace	Option + Delete

Change the default Search preferences for all users

Individual users can change the default Search preferences for syntax highlighting, auto-formatting, and line numbering features for themselves.

The default Search preferences can also be changed globally for all users.

Prerequisites

- Only users with file system access, such as system administrators, can change the default Search preferences for all users. If you are using Splunk Cloud and want to change the default Search settings for your Splunk system, open a Support ticket.

- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

Steps

1. Open the local `user-prefs.conf.spec.in` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Under the [general] stanza, you can change the settings listed in the following table.

Feature	Attribute syntax	Default setting
Syntax highlighting	search_syntax_highlighting = <boolean>	true
Auto-formatting	search_auto_format = <boolean>	false
Line numbering	search_line_numbers = <boolean>	false

3. Restart the Splunk instance.

Search actions

Splunk software provides a set of controls that you can use to manage "in process" searches and to create reports and dashboards.

Control search job progress

After you launch a search, you can access and manage information about the search **job** without leaving the Search view.

1. After your search is running, paused, or finalized, click **Job** from the Search actions group.

2. Select an option from the list.
    - **Edit job settings.** Opens the Job Settings dialog, where you can change the read permissions for the job, extend the job lifespan, and get a URL for the job. You can use the URL to share the job with others or to add a bookmark to the job in your Web browser.
    - **Send job to the background.** Runs the job on the background. Use this option if the search job is slow to complete. This enables you to work on other activities, including running a new search job.
    - **Inspect job.** Opens the Search Job Inspector window and displays information and metrics about the search job. You can select this action while the search is running or after the search completes. For more information, see View search job properties.
    - **Delete job.** Deletes the current job, even if that job is running, paused, or has finalized. After you delete the job you can still save the search as a report.

For more information, see About jobs and job management.

Change the search mode

The search mode controls the search experience. The default search mode is Smart Mode.

Fast Mode
: Speeds up searches by cutting down on the amount of event information that the search returns.

Verbose Mode
: Returns as much event information as possible.

Smart Mode
: Automatically toggles the search behavior between Fast Mode and Verbose Mode, based on the type of search that you are running.

For more information, see Search modes in this manual.

Save the results

The **Save as** menu lists options for saving the results of a search as a report, dashboard panel, alert, and event type.

Report
: Saves a search as a **report** to use the search again later. You can run the report again from the Reports page. You access the Reports page from the App bar. Read more about how to Create and edit reports in the *Reporting Manual*.

Dashboard Panel
: Generates a dashboard **panel** based on your search and add it to a new or existing dashboard. To learn more, see the Dashboard overview in the *Dashboards and Visualizations* manual.

Alert
: Defines an **alert** based on your search. An alert runs a report in the background (either on a **schedule** or in **real time**). When the search returns results that meet a condition you have set in the alert definition, the alert is triggered. For more information, see the *Alerting Manual*.

Event Type
: Classify events that have common characteristics. If the search does not include a **pipe operator** or a **subsearch**, you can use this option to save the search as an event type. For more information, see About event types and Define event types in Splunk Web in the *Knowledge Manager* manual.

Other search actions

Between the job progress controls and search mode selector are three buttons which enable you to **Share**, **Export**, and **Print** the results of a search.

- Click **Share** to share the job. When you select this, the job's lifetime is extended to 7 days and read permissions are set to Everyone. For more information about jobs, see About jobs and job management in this manual.
- Click **Export** to export the results. You can select to output to CSV, raw events, XML, or JSON and specify the number of results to export.
- Click **Print** to send the results to a printer that has been configured.

Additionally, use the **Close** button next to **Save as** menu to cancel the search and return to Splunk Home.

Search modes

You can use the Search Mode selector to provide a search experience that fits your needs.

The search mode selector is on the right side of the Search bar. The modes are Smart, Fast, and Verbose. The default mode is Smart.

Depending on the mode you set, you can see all the data available for your search but at the expense of longer search times, or you can speed up and streamline your search in certain ways.

The Fast and Verbose modes represent the two ends of the search mode spectrum. The default Smart mode switches between the Fast and Verbose modes depending on the type of search that you are running. When you first run a saved search, it runs in the Smart mode.

Using the Fast mode

The Fast mode prioritizes the performance of the search and does not return nonessential field or event data. This means that the search returns what is essential and required.

- **Disables field discovery. Field discovery** is the process Splunk software uses to extract fields aside from **default fields** such as `host`, `source`, and `sourcetype`. The Splunk software only returns information on default fields and fields that are required to fulfill your search. If you are searching on specific fields, those fields are extracted.
- **Only depicts search results as report result tables or visualizations when you run a reporting search**. A reporting search is a search that includes **transforming commands**. Under the Fast mode you will see only event lists and event timelines for searches that *do not* include

transforming commands.

For more information about what the Splunk software does when field discovery is enabled or disabled, see When Splunk Enterprise extracts fields in the *Knowledge Manager Manual*.

Using the Verbose mode

The Verbose mode returns all of the field and event data it possibly can, even if it means the search takes longer to complete, and even if the search includes reporting commands.

- **Discovers all of the fields it can.** This includes default fields, automatic search-time field extractions, and all user-defined index-time and search-time field extractions. Discovered fields are displayed in the left-hand fields sidebar in the Events results tab.
- **Returns an event list view of results and generates the search timeline.** It also generates report tables and visualizations if your search includes reporting commands.

You may want to use the Verbose mode if you are putting together a transforming search but are not exactly sure what fields you need to report on, or if you need to verify that you are summarizing the correct events.

Reports cannot benefit from report acceleration when you run them in Verbose mode. If you enable report acceleration for a report and it has been running faster as a result, be aware that if you switch the mode of the search to Verbose it will run at a slower, non-accelerated pace.

Report acceleration is designed to be used with slow-completing searches that have over 100k events and which utilize **transforming commands**. For more information see Accelerate reports, in the *Reporting Manual*.

Using the Smart mode

All reports run in Smart mode, the default search mode, after they are first created. By design, the Smart mode returns the best results for whatever search or report you run. If you search on events, you get all the event information you need. If your run a transforming search, the Splunk software favors speed over thoroughness and brings you straight to the report result table or visualization.

When you run a Smart mode search that *does not* include transforming commands, the search behaves as if it were in Verbose mode.

- **Discovers all the fields it can.**
- **Generates the full event list and event timeline.** No event table or visualization will appear because you need transforming commands to make those happen.

When you run a Smart mode search that includes transforming commands, the search behaves as if it were in Fast mode.

- **Disables field discovery.**
- **Does not waste time generating the event list and event timeline** and jumps you straight to the report result table or visualization.

For more information about transforming commands and transforming searches, see About reporting commands in the *Search Manual*.

Search history

There are several ways to view your search history.

Search history in the Search Summary view

Your full search history appears at the bottom of the Search Summary view.

Use the **Search History** panel to view and interact with the searches that you have run previously.

Click **Expand your search history** to view your search history. The search history displays as a table with the following columns:

Search
: Contains the search string, displayed as plain text so that you can copy the contents. By default, the Search History table truncates the search string to fit on a single line. For longer search strings, you can click the expand icon to the left of the search string to display the full search string.

Actions
: Contains the action, **Add to search**. Click **Add to Search** to replace the contents of the search bar with the selected historical search contents. You can use keyboard shortcuts to display the search in a new browser tab.
Windows: Use CTRL+ click on **Add to Search**

Mac: Use Command + click on **Add to Search**

Last Run
: Contains the date and time when the search was last run.

Filter to locate searches

You can filter your search history to quickly find the search that you are looking for. You can filter by keyword or filter by time.

- Type keywords into the **filter** text box to locate historical searches that contain the keyword. For example, type **sourcetype=access_*** to locate the searches that contain this criteria.
- Select from the list of time filters based on when the search was last run. Select either **Today, Last 7 Days**, or **Last 30 Days**. To see the entire search history, select **No Time Filter**.

Sort search history

In the Search History table, click the **Search** column header to sort the searches alphabetically by search criteria. Click the **Last Run** column heading to sort the searches by the date that the search was run. You can sort the list in ascending or descending order by clicking the column heading again.

Change the page display

By default, the search history shows your most recent 20 searches. Subsequent pages show older searches. Click **20 Per Page** to change how many searches appear on each page in the search history list. Choose from 10, 20, or 50 searches to display on each page.

Search history with the Search Assistant

Additionally, as you type search criteria into the Search bar, the Search Assistant shows searches from your history as a possible **Matching Search** to the criteria you are typing in.

See Also

Navigating Splunk Web

Search Primer

Search command primer

At the beginning of a search pipeline, the `search` command is implied, even though you do not explicitly specify it. If you type in

```
host=webserver*
```

It is as if you typed in

```
search host=webserver*
```

Use keywords, phrases, fields, boolean expressions, and comparison expressions to specify exactly which events you want to retrieve from Splunk indexes.

For specific information see:

- Wildcards
- Boolean expressions
- Field expressions
- Difference between NOT and !=
- Use CASE and TERM to match phrases
- SPL and regular expressions

Keywords and phrases

By default, when you search with keywords and phrases, Splunk software retrieves events by matching against the raw event field, `_raw`, in your data. When you start adding search modifiers, such as fields like `_time` and `tag`, you are also matching against pieces of information that have been extracted from the `_raw` field.

When searching for strings, which includes keywords and quoted phrases (or anything that's not a search modifier), Splunk software searches the `_raw` field for the matching events or results. Some examples of keywords and phrases are:

```
web
```

```
error
```

```
web error
```

```
"web error"
```

Note that the search for the quoted phrase "web error" is not the same as the search before it. When you search for `web error`, Splunk software returns events that contain both "web" and "error". When you search for "web error", Splunk software only returns events that contain the phrase "web error".

See Also

- Understanding SPL syntax
- About retrieving events
- About search time ranges
- Quick tips for optimization

Wildcards

Use the asterisk wildcard (*) character to match an unrestricted number of characters in a string. If you specify an asterisk with no other criteria, you are asking to match everything. All events are retrieved up to the maximum limit. Searching for `*` as part of a string, generates matches based on that string. For example:

- `my*` matches `myhost1`, `myhost.ny.mydomain.com`, `myeventtype`, and so on.
- `*host` matches `myhost`, `yourhost`, and so on.
- `*host*` matches `host1`, `myhost3`, `yourhost27.yourdomain.com`, and so on.

The more specific your search terms are to the events that you want to retrieve, the better chance you have of matching the terms. For example, searching for `access denied` is always better than searching for `denied`. If 90% of your events have the word `error` but only 5% have the word `sshd`, and the events that you want to find require both of these words, include `sshd` in the search to make it more efficient.

When to avoid wildcard characters

There are several situations in which you should avoid using wildcard characters.

Avoid using wildcards in the middle of a string

Wildcard characters in the middle of a word or string might cause inconsistent results. This is especially true if the string contains punctuation, such as an underscore _ or dash - character.

For example, suppose you have the following list of product IDs.

```
DB-SG-G01
DC-SG-G02
MB-AG-G07
MB-AG-T01
SC-MG-G01
SF-BVS-G01
SG-SH-G05
WC-SH-A02
WC-SH-G04
```

You create a search that looks for all of the product IDs that begin with the letter S and end in G01.

```
...productID=S*G01 ...
```

Because the product IDs contain punctuation, the search results might be inconsistent because of the way in which data that contains punctuation is indexed.

If the number of product IDs is small, specify the exact product IDs in your search. For example:

```
...productID=SC-MG-G01 OR productID=SF-BVS-G01 ...
```

If the number of product IDs is large, use a lookup. See About lookups and work flow actions.

Avoid using wildcards to match punctuation

Punctuation are characters that are not numbers or letters. If you want to match part of a string that includes punctuation, specify each string with the punctuation that you are searching for.

For example, you have the following values in the `uri_path` field in your events.

```
/cart.do
/cart/error.do
/cart/success.do
/category.screen
/oldlink
/product.screen
/productscreen.html
/show.do
/stuff/logo.ico
```

You want to match every `uri_path` that starts with `/cart`. The problem is that the paths contain a forward slash (/) character and period (.) character. Instead of specifying a wildcard character for the punctuation such as `/cart*`, specify the punctuation directly in your search criteria. For example, specify `/cart.do OR /cart/error.do OR /cart/success.do`.

Prefix wildcards might cause performance issues

When you use a wildcard character at the beginning of a string, performance degradation might occur.

Boolean expressions

The Splunk search processing language (SPL) supports the Boolean operators: `AND`, `OR`, and `NOT`.

The operators must be capitalized.

The AND operator is always implied between terms, that is: `web error` is the same as `web AND error`. So unless you want to include it for clarity reasons, you should not need to specify the AND operator.

The NOT operator only applies to the term immediately following NOT. To apply to multiple terms, you must enclose the terms in parenthesis.

Inclusion is generally better than exclusion. Searching for "access denied" will yield faster results than NOT "access granted".

Order of evaluation

The order in which the Splunk software evaluates Boolean expressions depends on whether you are using the expression with the `search` command or the `where` command. This includes the implied `search` command at the beginning of the search.

The following table describes the order in which the Boolean expressions are evaluated.

Order	Search command	Where command
1	Expressions within parentheses	Expressions within parentheses
2	`NOT` clauses	`NOT` clauses
3	`OR` clauses	`AND` clauses
4	`AND` clauses	`OR` clauses

Examples

The following examples show how Splunk software processes Boolean expressions.

Consider the following search:

```
A=1 AND B=2 OR C=3
```

This is the same as specifying `A=1 B=2 OR C=3`

When you specify values without parenthesis, this search is processed as:

```
A=1 AND ( B=2 OR C=3 )
```

Here is another example:

```
error NOT 403 OR 404
```

Without parenthesis, this search is processed as:

- Search for any event that contains the string "error" and does not contain the keyword 403
- Search for any event that contains the string "error" and 404

You can use parentheses to group Boolean expressions. For example:

```
error NOT (403 OR 404)
```

```
(A=1 AND B=2 ) OR C=3
```

Field expressions

When you add data, Splunk software extracts pairs of information and saves them as fields. Some fields are common to all events, but others are not. Adding fields to your search term gives you a better chance of matching specific events.

If you are searching web access logs for specific HTTP status errors, instead of searching for "web error 404", you can use fields to search for:

```
status=404
```

See Use fields to retrieve events.

Use comparison operators to match field values

You can use comparison operators to match a specific value or a range of field values.

Operator	Example	Result
=	field=foo	Multivalued field values that exactly match "foo".
!=	field!=foo	Multivalued field values that don't exactly match "foo".
<	field<x	Numerical field values that are less than x.
>	field>x	Numerical field values that are greater than x.
<=	field<=x	Numerical field values that are less than and equal to x.
>=	field>=x	Numerical field values that are greater than and equal to x.

For example, to find events that have a delay field that is greater than 10:

```
delay > 10
```

Difference between NOT and !=

When you want to exclude results from your search you can use the NOT operator or the `!=` field expression. However there is a significant difference in

the results that are returned from these two methods.

Suppose you have the following events. As you can see, some events have missing values.

ID	Name	Color	Location
101M3	McIntosh	Chestnut	Marin Meadows
104F5	Lyra	Bay	
104M6	Rutherford	Dun	Placer Pastures
101F2	Rarity		Marin Meadows
102M7	Dash	Black	Calaveras Farms
102M1		Roan	
101F6		Chestnut	Marin Meadows
104F4	Pinkie	Sorrel	Placer Pastures
102M8	Spike	Grey	Calaveras Farms

Searching with !=

If you search with the `!=` expression, every event that has a value in the field, where that values does not match the value you specify, is returned. Events that do not have a value in the field are not included in the results.

For example, if you search for `Location!="Calaveras Farms"`, events that do not have Calaveras Farms as the Location are returned. Events that do not have Location value are not included in the results.

```
source="Ponies.csv" Location!="Calaveras Farms"
```

ID	Name	Color	Location
101M3	McIntosh	Chestnut	Marin Meadows
104M6	Rutherford	Dun	Placer Pastures
101F2	Rarity		Marin Meadows
101F6		Chestnut	Marin Meadows
104F4	Pinkie	Sorrel	Placer Pastures

If you search for a Location that does not exist using the `!=` expression, all of the events that have a Location value are returned.

Searching with NOT

If you search with the NOT operator, every event is returned except the events that contain the value you specify. This includes events that do not have a value in the field.

For example, if you search using `NOT Location="Calaveras Farms"`, every event is returned except the events that contain the value "Calaveras Farms". This includes events that do not have a Location value.

```
source="Ponies.csv" NOT Location="Calaveras Farms"
```

ID	Name	Color	Location
101M3	McIntosh	Chestnut	Marin Meadows
104F5	Lyra	Bay	
104M6	Rutherford	Dun	Placer Pastures
101F2	Rarity		Marin Meadows
102M1		Roan	
101F6		Chestnut	Marin Meadows
104F4	Pinkie	Sorrel	Placer Pastures

If you search for a Location that does not exist using NOT operator, all of the events are returned.

Searching with != or NOT is not efficient

Using the `!=` expression or NOT operator to exclude events from your search results is not an efficient method of filtering events. The execution cost for a search is actually less when you explicitly specify the values that you want to include in the search results. For more tips on search optimization, see Quick tips for optimization.

Use CASE() and TERM() to match phrases

If you want to search for a specific term or phrase in your Splunk index, use the CASE() or TERM() directives to do an exact match of the entire term.

- CASE: Search for case-sensitive matches for terms and field values.

- TERM: Match whatever is inside the parentheses as a single term in the index, even if it contains characters that are usually recognized as minor segmenters, such as periods or underscores.

When you search for a term that contains minor segmenters, it is treated by default as a phrase: It searches for the conjunction of the subterms (the terms between minor breaks) and post-filters the results. For example, when you search for the IP address 127.0.0.1, Splunk software searches for: `127 AND 0 AND 1`

This search is not very efficient if the conjunction of these subterms is common, even if the whole term itself is not common.

If you search for TERM(127.0.0.1), Splunk software treats the IP address as a single term to match in your raw data.

TERM is more useful for cases where the term contains minor segmenters and is bounded by major segmenters, such as spaces or commas. In fact, TERM does not work for terms that are not bounded by major breakers. This is illustrated in the examples below.

For more information about how Splunk software breaks events up into searchable segments, read About segmentation in *Getting Data In*.

Examples

TERM(127.0.0.1) works for raw data that looks like:

127.0.0.1 - admin

However, it fails for data that looks like:

ip=127.0.0.1 - user=admin

This is because "=" is a minor breaker and the IP address portion of the event is indexed as: ip, 127, 0, 1, ip=127.0.0.1

If your data looks like this:

ip 127.0.0.1 - user admin

TERM(user admin) fails to return results. The space is a major breaker and the phrase "user admin" is not indexed as a single term.

SPL and regular expressions

Splunk regular expressions are PCRE (Perl Compatible Regular Expressions).

You can use regular expressions with the `rex` and `regex` commands. You can also use regular expressions with evaluation functions such as `match` and `replace`.

Here are a few things that you should know about using regular expressions in Splunk searches.

Pipe characters

A pipe character (|) is used in regular expressions to specify an OR condition. For example, A or B is expressed as A | B.

Because pipe characters are used to separate commands in SPL, you must enclose a regular expression that uses the pipe character in quotation marks. For example:

```
...|regex "expression | with pipe"
```

This is interpreted by SPL as a search for the text "expression" OR "with pipe".

Backslash characters

The backslash character (\ ) is used in regular expressions to "escape" special characters. For example. The period character is used in a regular expression to match any character, except a line break character. If you want to match a period character, you must escape the period character by specifying `\.` in your regular expression.

Splunk SPL uses the asterisk (*) as a wildcard character. The backslash cannot be used to escape the asterisk in search strings.

Searches that include a regular expression that contains a double backslash encounters a double backslash, such as in a filepath like `c:\\temp`, the search interprets the first backslash as a regular expression escape character. The

filepath is interpreted as `c:\temp`, one of the backslashes is removed.

You must escape both backslash characters in a filepath by specifying **4** consecutive backslashes for the root portion of the filepath. For example: `c:\\\\temp`. For a longer filepath, such as `c:\\temp\example`, you would specify `c:\\\\temp\\example` in your regular expression in the search string.

More about regular expressions

For more information:

- See Extract fields using regular expressions
- See About Splunk regular expressions in the *Knowledge Manager Manual*.

Optimizing Searches

About search optimization

Search optimization is a technique for making your search run as efficiently as possible.

When not optimized, a search often runs longer, retrieves larger amounts of data from the indexes than is needed, and inefficiently uses more memory and network resources. Multiply these issues by hundreds or thousands of searches and the end result is a slow or sluggish system.

There are a set of basic principles that you can follow to optimize your searches.

- Retrieve only the required data
- Move as little data as possible
- Parallelize as much work as possible
- Set appropriate time windows

To implement the search optimization principles, use the following techniques.

- Filter as much as possible in the initial search
- Perform joins and lookups on only the required data
- Perform evaluations on the minimum number of events possible
- Move commands that bring data to the search head as late as possible in your search criteria

Indexes and searches

When you run a search, the Splunk software uses the information in the index files to identify which events to retrieve from disk. The smaller the number of events to retrieve from disk, the faster the search runs.

How you construct your search has a significant impact on the number of events retrieved from disk.

When data is indexed, the data is processed into events based on time. The processed data consists of several files:

- The raw data in compressed form (**rawdata**)
- The indexes that point to the raw data (**index files**, also referred to as **tsidx files**)
- Some metadata files

These files are written to disk and reside in sets of directories, organized by age, called **buckets**.

Use indexes effectively

One method to limit the data that is pulled off from disk is to partition data into separate indexes. If you rarely search across more than one type of data at a time, partition different types of data into separate indexes. Then restrict your searches to the specific index. For example, store web access data in one index and firewall data in another. Using separate indexes is recommended for sparse data, which might otherwise be buried in a large volume of unrelated data.

- See Ways to set up multiple indexes in the *Managing Indexers and Clusters of Indexers* manual
- See Retrieve events from indexes

A tale of two searches

Some frequently used searches unnecessarily consume a significant amount of system resources. You will learn how optimizing just one search can save significant system resources.

A frequently used search

One search that is frequently used is a search that contains a lookup and an evaluation, followed by another search. For example:

```
sourcetype=my_source | lookup my_lookup_file D OUTPUTNEW L | eval E=L/T
| search A=25 L>100 E>50
```

The following diagram shows a simplified, visual representation of this search.

When the search is run, the index is accessed and 1 million events are extracted based on the source type.

In the next part of the search, the `lookup` and `eval` command are run are on all 1 million events. Both the `lookup` and `eval` commands add columns to the events, as shown in the following image.

Finally, a second search command runs against the columns A, L, and E.

- For column A, the search looks for values that are equal to 25.
- For column L, which was added as a result of the `lookup` command, the search looks for values greater than 100.
- For column E, which was added as a result of the `eval` command, the search looks for values that are greater than 50.

Events that match the criteria for columns A, L, and E are identified, and 50,000 events that match the search criteria are returned. The following image shows the entire process and the resource costs involved in this inefficient search.

An optimized search

You can optimize the entire search by moving some of the components from the second `search` to locations earlier in the search process.

Moving the criteria `A=25` before the first pipe filters the events earlier and reduces the amount of times that the index is accessed. The number of events extracted is 300,000. This is a reduction of 700,000 compared to the original search. The `lookup` is performed on 300,000 events instead of 1 million events.

Moving the criteria `L>100` immediately after the `lookup` filters the events further, reducing the number of events returned by 100,000. The `eval` is performed on 200,000 events instead of 1 million events.

The criteria `E>50` is dependent on the results of the `eval` command and cannot be moved. The results are the same as the original search. 50,000 events are returned, but with much less impact on resources.

This is the optimized search.

```
sourcetype=my_source A=25 | lookup my_lookup_file D OUTPUTNEW L |
search L>100 | eval E=L/T | search E>50
```

The following image shows the impact of rearranging the search criteria.

See also

- Quick tips for optimization
- Write better searches
- Built-in optimizations

Quick tips for optimization

The key to fast searching is to limit the data that needs to be pulled from disk to an absolute minimum. Filter the data as early as possible in the search, so that processing is done on the minimum amount of data necessary.

Limit the data from disk

The techniques to limit the amount of data retrieved from disk range from setting a narrow time window, being as specific as possible, and retrieving the smallest number of events necessary.

Narrow the time window

One of the most effective ways to limit the data that is pulled off from disk is to limit the time range. Use the time range picker or specify time modifiers in your search to identify the smallest window of time necessary for your search.

If you need to see data from only the last hour, do not use the default time range of **Last 24 hours**.

- See Select time ranges to apply to your search
- See Specify time modifiers in your search

If you must use a broad time range, such as **Last week** or **All time**, then use other techniques to limit the amount of data retrieved from disk.

Specify the index, source, or source type

Understanding how your data is organized is important to optimizing your searches. Take the time to learn which indexes contain your data, the sources of your data, and the source types. Knowing this information about your data helps you narrow down your searches.

1. Run the following search.

   ```
   source=*
   ```

   This search not optimized, but it does provide an opportunity for you to learn about the data you have access to.
2. In the **Selected fields** list, click on each type of field and look at the values for host, source, and sourcetype.
3. In the **Interesting fields** list, click on the **index** field. Look at the names of the indexes that you have access to.

Whenever possible, specify the index, source, or source type in your search. When Splunk software indexes data, it automatically tags each event with a number of fields. The index, source, and source type fields are added automatically to each event as default fields. A *default field* is an indexed field that the Splunk software recognizes in your event data at search time. The host, source, and source type fields describe where the event originated.

Be specific

Use the most specific terms in your search that you can. If possible, avoid using wildcard characters.

> For example, instead of using a wildcard character for a keyword:
> `*error`
> Use the specific keyword:
> `fatal_error`

Here is another example.

> Instead of using a wildcard character for field values:
> `status=404 OR status=5*`

Specify each value:
`status=404 OR status=500 OR status=503`

Combine a source type or an index with one or more field-value pairs. For example:

```
sourcetype=access_* status=200 action=purchase
```

This search retrieves events from only your web access logs. A wildcard character, access_*, is used in the field value to match any Apache web access source type. The source types can be `access_common`, `access_combined`, or `access_combined_wcookie`. Two specific field-value pairs are included in the search, `status=200` and `action=purchase`.

Limit the number of events retrieved

You can specify a limit to the number of events retrieved by using the `head` command. The `head` command retrieves only the most recent N events for a historical search, or the first N captured events for a realtime search.

Limiting the number of events retrieved is useful in several situations:

- You are creating a search and want to determine if you are retrieving the correct events
- You need only a subset or sample set of events for your search

For example:

```
sourcetype=access_* | head 1000 ...
```

Avoid using NOT expressions

More resources are used tracking NOT expressions than if you specify what you are looking for. Where ever possible, avoid using NOT expressions. For example, instead of using a string of NOT or != expressions such as:

```
(NOT host=d NOT host=e)
```

or

```
(host!=d AND host!=e)
```

Use the specific terms you are searching for:

`(host=a OR host=b OR host=c).`

To learn more, see Difference between NOT and !=.

Filter as soon as possible

Filter results as soon as possible before performing calculations. You can use field-value pairs and commands to filter results.

Use field-value pairs before the first pipe

Field-value pairs are indexed. Specifying field-value pairs before the first pipe is an efficient way to filter out events.

> For example, in the following search the term `status=404` is in a separate search:
> `ERROR | search status=404`
> Move the term `status=404` before the first pipe:
> `ERROR status=404`

Here is another example.

> The second search includes the term `clientip="10.0.0.0/8"`. There is no reason to wait to filter on that term.
> `ERROR | stats sum(bytes) as sum by clientip | search sum >1048576 AND clientip="10.0.0.0/8"`
> Move the term `clientip="10.0.0.0/8"` to filter out all other clientip addresses before the `stats` command.
> `ERROR clientip="10.0.0.0/8" | stats sum(bytes) by clientip | search sum > 1048576`

Use filtering commands before calculating commands

Use filtering commands, such as `where`, before commands that perform calculations, such as `eval`.

> For example, this search has a `where` command after the `eval` command. The search does not require the results of the `eval` command before the `where` command is run.
> `field1=value | eval KB=bytes/1024 | where field2=field3`
> Move the `where` command to filter the results before the `eval` command is processed:
> `field1=value | where field2=field3 | eval KB=bytes/1024`

Filter unnecessary fields from search results

You can remove unnecessary fields from the search results by using commands such as `fields`.

Use non-streaming commands as late as possible

Postpone commands like `dedup`, `sort`, and `stats` as late as possible in your search. These commands are referred to as non-streaming commands. Before these commands can run, the entire result set must be returned. For example, the results cannot be sorted until all of the results are available.

> For an explanation about the differences between streaming and non-streaming commands, see Types of commands.
>
> For a list of of commands by type, see Command types in the *Search Reference*.

Other techniques for search optimization

There are a few other techniques that you can use to optimize your searches.

- Use post-process searches in dashboards. See Searches power dashboards and forms in *Dashboards and Visualizations*.
- Use **summary indexing**, **report acceleration**, and **data model acceleration** features.
- Use **Fast Mode** to increase the speed of searches by reducing the event data that the search returns. See Search modes.

See also

- About search optimization
- Write better searches
- Built-in optimizations

Write better searches

This topic examines some causes of slow searches and includes guidelines to help you write searches that run more efficiently. Many factors can affect the

speed of your searches, including:

- The volume of data that you are searching
- How your searches are constructed
- The number of concurrent searches

To optimize the speed at which your search runs, minimize the amount of processing time required each component of the search.

Know your type of search

The recommendations for optimizing searches depend on the type of search that you run and the characteristics of the data you are searching. Searches fall into two types, that are based on the goal you want to accomplish. Either a search is designed to retrieve events or a search is designed to generate a report that summarizes or organizes the data.

See Types of searches.

Searches that retrieve events

Raw event searches retrieve events from a Splunk index without any additional processing of the events that are retrieved. When retrieving events from the index, be specific about the events that you want to retrieve. You can do this with keywords and field-value pairs that are unique to the events.

If the events you want to retrieve occur frequently in the dataset, the search is called a *dense search*. If the events you want to retrieve are rare in the dataset, the search is called a *sparse search*. Sparse searches that run against large volumes of data take longer than dense searches against the same data set.

See Use fields to retrieve events.

Searches that generate reports

Report-generating searches, or transforming searches, perform additional processing on events after the events are retrieved from an index. This processing can include filtering, transforming, and other operations using one or more statistical functions against the set of results. Because this processing occurs in memory, the more restrictive and specific you are retrieving the events, the faster the search will run.

See About transformating commands and searches.

Command types and parallel processing

Some commands process events in a stream. There is one event in and one, or no, event out. These are referred to as **streaming commands**. Examples of streaming commands are `where`, `eval`, `lookup`, and `search`.

Other commands require all of the events from all of the indexers before the command can finish. These are referred to as **non-streaming commands**. Examples of non-streaming commands are `stats`, `sort`, `dedup`, `top`, and `append`.

Non-streaming commands can run only when all of the data is available. To process non-streaming commands, all of the search results from the indexers are sent to the search head. When this happens, all further processing must be performed by the search head, rather than in parallel on the indexers.

Parallel processing example

Non-streaming commands that are early in your search reduce parallel processing.

For example, the following image shows a search that a user has run. The search starts with the `search` command, which is implied as the first command in the Search bar. The search continues with the `lookup`, `where`, and `eval` commands. The search then contains a `sort`, based on the `Name` field, followed by another `where` command.

The search is sent to the search head and distributed to the indexers to process as much of the search as possible on the indexers.

For the events that are on each indexer, the indexer process the search until the indexer encounters a non-streaming command. In this example, the indexers process the search through the `eval` command. To perform the `sort`, all of the results must be sent to the search head for processing.

However, the results that are on each indexer can be sorted by the indexer. This is referred to as a *presort*. In this example the sort is on the `Name` field. The following image shows that the first indexer returns the names Alex and Maria. The second indexer returns the name Wei. The third indexer returns the names Claudia, David, and Eduardo.

To return the full list of results sorted by name, all of the events that match the search criteria must be sent to the search head. When all of the results are on the search head, the rest of the search must be processed on the search head. In this example the `sort` and any remaining commands are processed on the search head.

The following image shows that each indexer has presorted the results, based on the `Name` field. The results are sent to the search head, and are appended together. The search head then sorts the entire list into the correct order. The search head processes the remaining commands in the search to produce the final results. In this example, that includes the second `where` command. The final results are returned to the user.

When part or all of a search is run on the indexers, the search processes in parallel and search performance is much quicker.

To optimize your searches, place non-streaming commands as late as possible in your search string.

Tips for tuning your searches

In most cases, your search is slow because of the complexity of your query to retrieve events from index. For example, if your search contains extremely large OR lists, complex subsearches (which break down into OR lists), and types of phrase searches, it takes longer to process. This section discusses some tips for tuning your searches so that they are more efficient.

Performing statistics with a BY clause on a set of field values that have a high cardinality, lots of uncommon or unique values, requires a lot of memory. One possible remedy is to decrease the value for the `chunk_size` setting used with the `tstats` command. Additionally, reducing the number of distinct values that the BY clause must process can also be beneficial.

Restrict searches to the specific index

If you rarely search across more than one type of data at a time, partition your different types of data into separate indexes. Then restrict your searches to the specific index. For example, store Web access data in one index and firewall data in another. This is recommended for sparse data, which might otherwise be buried in a large volume of unrelated data.

See Create custom indexes in *Managing Indexers and Clusters of Indexers* and Retrieve events from indexes in this manual.

Use fields effectively

Searches with fields are faster when they use fields that have already been extracted (indexed fields) instead of fields extracted at search time. For more information about indexed fields and default fields, see About fields in the *Knowledge Manager Manual*.

Use indexed and default fields

Use indexed and default fields whenever you can to help search or filter your data efficiently. At index time, Splunk software extracts a set of default fields that are common to each event. These fields include `host`, `source`, and `sourcetype`. Use these fields to filter your data as early as possible in the search so that processing is done on a minimum amount of data.

For example, if you're building a report on web access errors, search for those specific errors before the reporting command:

```
sourcetype=access_* (status=4* OR status=5*) | stats count by status
```

Specify indexed fields with <field>::<value>

You can also run efficient searches for fields that have been indexed from structured data such as CSV files and JSON data sources. When you do this, replace the equal sign with double colons, like this: `<field>::<value>`.

This syntax works best in searches for fields that have been indexed from structured data, though it can be used to search for default and custom indexed fields as well. You cannot use it to search on **Search-time** fields.

Disable field discovery to improve search performance

If you don't need additional fields in your search, set **Search Mode** to a setting that disables field discovery to improve search performance in the timeline view or use the `fields` command to specify only the fields that you want to see in your results.

However, disabling field discovery prevents automatic **field extraction**, except for fields that are required to fulfill your search, such as fields that you are specifically searching on and **default fields** such as `_time`, `host`, `source`, and `sourcetype`. The search runs faster because Splunk software is no longer trying to extract every field possible from your events.

Search mode is set to **Smart** by default. Set the search mode to **Verbose** if you are running searches with reporting commands, you don't know what fields exist in your data, and think you might need the fields to help you narrow down your search.

See Set search modes.

Also see the topic about the fields command in the *Search Reference*.

Summarize your data

It can take a lot of time to search through very large data sets. If you regularly generate reports on large volumes of data, use summary indexing to pre-calculate the values that you use most often in your reports. Schedule saved searches to collect metrics on a regular basis, and report on the summarized data instead of on raw data.

See Use summary indexing for increased reporting efficiency.

Use the Search Job Inspector

The **Search Job Inspector** is a tool you can use both to troubleshoot the performance of a search and to determine which phase of the search takes the greatest amounts of time. It dissects the behavior of your searches to help you understand the execution costs of knowledge objects such as event types, tags, lookups, search commands, and other components within the search.

See View search job properties in this manual.

See also

- About search optimization
- Quick tips for optimization
- Built-in optimizations

Built-in optimization

The Splunk software includes built-in optimizations that analyze and process your searches for maximum efficiency.

One goal of these optimizations is to filter results as early as possible. Filtering reduces the amount of data to process for the search. Early filtering avoids unnecessary lookups and evaluation calculations for events that are not part of the final search results.

Another goal is to process as much as possible in parallel on the indexers. The built-in optimizations can reorder search processing, so that as many commands as possible are run in parallel on the indexers before sending the search results to the search head for final processing. You can see the reordered search using the Job Inspector. See Analyze search optimizations.

Predicate optimization

Predicates act as filters to remove unnecessary events as your search is processed. The earlier in the search pipeline that a predicate is applied, the more efficient is the use of system resources. If a predicate is applied early, when events are fetched, only the events that match the predicate are fetched. If the same predicate is applied later in the search, then resources are wasted fetching more events than are necessary.

There are several built-in optimizations that focus on predicates:

- Predicate merge
- Predicate pushdown
- Predicate splitting

Types of predicates

There are two types of predicates, simple predicates and complex predicates.

- A simple predicate is a single conditional of the form `field = value`.
- A complex predicate is a combination of several conjunctions (AND and OR) and disjunctions (NOT). For example, `Field1 = Value1 OR Field2 = Value2 AND Field3 = Value3`.

A complex predicate can be merged or split apart to optimize a search.

In Splunk SPL, there are two commands that perform predicate-based filtering, `where` and `search`.

An example of using the `where` command for filtering is:

```
index="_internal" | where bytes > 10
```

An example of using the `search` command for filtering is:

```
index="_internal" | search bytes > 10
```

Search-based predicates are a subset of where-based predicates. In other words, search-based predicates can be replaced by where-based predicates. However, where-based predicates cannot be replaced by search-based predicates.

Predicate merge

The predicate merge optimization takes two predicates and merges the predicates into one predicate. For example, consider the following search:

```
index=main | search a > 10 AND fieldA = "New"
```

There are two search commands in this example. The `search` command before the pipe, which is implied at the beginning of every search, and the explicit `search` command after the first pipe. With the predicate merge optimization, the predicates in the second `search` command are merged with the predicates in the first `search` command. For example:

```
index=main AND a > 10 AND fieldA = "New"
```

In some cases, predicates used with the `where` command can also be merged. For example, consider the following search:

```
index=main | where fieldA = "New"
```

With the predicate merge optimization, the predicates specified with the `where` command are merged with the predicates specified with the `search` command.

```
index=main AND fieldA = "New"
```

Issues with field extractions and predicate merge

An **inline field extraction** requires special handling if the regular expression pattern extracts a subtoken. The field must be set to `indexed=false` in the `fields.conf` file. See Example inline field extraction configurations in the *Knowledge Manager Manual*.

Consider the following sample event:

```
Mon Apr 17 16:08:16 2017 host=10.10.1.1  Login  name=John
 SUCCESS FRANCE
```

You create an extracted field called `country` that uses the following regular expression:

```
SUCCESS\s+?\S{3}
```

- `SUCCESS` matches the characters SUCCESS literally and is case sensitive.
- `\s` matches any whitespace character (space, tab, new line).
- `+?` is a quantifier that matches between one and unlimited times, the fewest needed to match the pattern.
- `\S` matches any non-whitespace character.
- `{3}` is a quantifier that matches exactly 3 non-whitespace characters

For the sample event, the following regular expression extracts the first 3 characters of the word FRANCE or `FRA`. The extraction `FRA` is a subtoken of the indexed term FRANCE.

When you search using the extracted field, for example:

```
index=main | search country=FRA
```

The search is optimized, using the predicate merge optimizer:

```
index=main country=FRA
```

However, the search returns no results because `FRA` is not part of the index. `FRANCE` is the indexed term.

To overcome this issue, you must add the following setting to the `fields.conf` file:

```
[country]
indexed=false
```

Alternatively, you can disable the built-in optimizations. See Optimization settings.

Predicate pushdown

The predicate pushdown optimization analyzes a search and reorders the search processing so that predicates are processed as early as possible.

Example of a simple predicate pushdown

You perform the following search:

```
sourcetype=access* (status=401 OR status=403) | lookup usertogroup user
OUTPUT group | where src_category="email_server"
```

The `sourcetype=access* (status=401 OR status=403)` portion of the search retrieves 50,000 events. The `lookup` is performed on all 50,000 events. Then the `where` command is applied and filters out events that are not `src_category="email_server"` . The result is that 45,000 events are discarded and 5,000 events are returned in the search results.

If the search criteria used with the `where` command is applied before the `lookup`, more events are filtered out of the results. Only 5,000 events are retrieved from disk before the `lookup` is performed.

With predicate pushdown, the search is reordered for processing. The `where` command is eliminated by moving the search criteria `src_category="email_server"` before the first pipe.

```
sourcetype=access* (status=401 OR status=403)
src_category="email_server" | lookup usertogroup user OUTPUT group
```

Example of a complex predicate pushdown

Consider the following search fragment:

```
... | eval x=if(isnull(x) OR x=="", "missing", x) | where x = "hello"
```

In this situation, the `eval` command is assigning a default value when a field is null or empty. This is a common pattern in Common Information Model (CIM) data models.

The built-in optimization reorganizes the search criteria before processing the search. The `where` command is moved before the `eval` command.

```
... | where x = "hello" | eval x=if(isnull(x) OR x=="", "missing", x)
```

Predicate splitting

Predicate splitting is the action of taking a predicate and dividing, or splitting, the predicate into smaller predicates. The predicate splitting optimizer can then move the smaller predicates, when possible, to an earlier place in the search.

Consider the following search:

```
index="_internal" component = "SearchProcess" | eval a = (x + y) |
where a > 200 AND x < 10
```

Because `field a` is generated as part of the `eval` command, it cannot be processed earlier in the search. However, `field x` exists in the events and can be processed earlier. The predicate splitting optimization separates the components of the `where` portion of the search. The search is reordered to process eligible components earlier.

```
index="_internal" component = "SearchProcess" | where x < 10 | eval a =
(x + y) | where a > 200
```

The portion of the `where` command `x < 10` is moved to before the `eval` command. This move reduces the number of results that the `eval` command must process.

Projection elimination

This optimization analyzes your search and determines if any of the generated fields specified in the search are not used to produce the search results. If

generated fields are identified that can be eliminated, an optimized version of the search is run. Your search syntax remains unchanged.

For example, consider the following search:

```
index=_internal | eval c = x * y / z | stats count BY a, b
```

The `c` field that is generated by `eval c = x * y / z` is not used in the `stats` calculation. The `c` field can be eliminated from the search.

Your search syntax remains:

```
index=_internal | eval c = x * y / z | stats count BY a, b
```

But the optimized search that is run is:

```
index=_internal | stats count BY a, b
```

Here is another example:

```
| tstats count FROM datamodel=internal_audit_logs
```

For buckets whose data models are not built, this expands to the following fallback search:

```
| search ((index=* OR index=_*) index=_audit)
| eval nodename = "Audit"
| eval is_searches=if(searchmatch("(action=search NOT
dmauditsearch)"),1,0),
  is_not_searches=1-is_searches,
is_modify=if(searchmatch("(action=edit_user
  OR action=edit_roles OR action=update)"),1,0),
is_not_modify=1-is_modify
| eval nodename = if(nodename == "Audit"
  AND searchmatch("(action=search NOT dmauditsearch)"),
mvappend(nodename,
  "Audit.searches"), nodename)
| eval is_realtime=case(is_realtime == 0, "false", is_realtime == 1,
"true",
  is_realtime == "N/A", "false"), search_id=replace(search_id,"'",""),
  search=replace(search,"'",""), search_type=case((id LIKE "DM_%"
  OR savedsearch_name LIKE "_ACCELERATE_DM%"), "dm_acceleration",
  search_id LIKE "scheduler%", "scheduled", search_id LIKE "rt%",
"realtime",
  search_id LIKE "subsearch%", "subsearch", (search_id LIKE
"SummaryDirector%"
  OR search_id LIKE "summarize_SummaryDirector%"), "summary_director",
```

```
  1=1, "adhoc")
| rename is_realtime AS Audit.searches.is_realtime
  search_id AS Audit.searches.search_id
  search AS Audit.searches.search
  search_type AS Audit.searches.search_type
| eval nodename = if(nodename == "Audit" AND
searchmatch("(action=edit_user
  OR action=edit_roles OR action=update)"), mvappend(nodename,
  "Audit.modify"), nodename)
| rename action AS Audit.action info AS Audit.info object AS
Audit.object
  operation AS Audit.operation path AS Audit.path user AS Audit.user
  exec_time AS Audit.exec_time result_count AS Audit.result_count
  savedsearch_name AS Audit.savedsearch_name
  scan_count AS Audit.scan_count total_run_time AS Audit.total_run_time
  is_searches AS Audit.is_searches is_not_searches AS
Audit.is_not_searches
  is_modify AS Audit.is_modify is_not_modify AS Audit.is_not_modify
| addinfo type=count label=prereport_events
| stats count
```

This search can be optimized to this syntax:

```
| search ((index=* OR index=_*) index=_audit) | stats count
```

Eliminating unnecessary generated fields, or projections, leads to better search performance.

Event types and tags optimization

Event typing and tagging can take a significant amount of search processing time. This is especially true with Splunk Enterprise Security where there are many event types and tags defined. The more event types and tags that are defined in the system, the greater the annotation costs.

The built-in event types and tags optimization applies to transforming searches and data model acceleration searches.

Transforming searches

When a transforming search is run without the built-in optimization, all of the event types and tags are uploaded from the configuration system to the search processor. This upload happens whether the search requires the event types and tags or not. The search processing attempts to apply the event types and tags to each event.

For example, consider the following search:

```
index=_internal | search tag=foo | stats count by host
```

This search needs only the tag `foo`. But without the optimization, all of the event types and tags are uploaded. The built-in optimization solves this problem by analyzing the search and only uploading the event types and tags that are required. If no event types or tags are needed, none are uploaded which improves search performance.

For transforming searches, this optimization is controlled by several settings in the `limits.conf` file, under the `[search_optimization::required_field_values]` stanza. These settings are turned on by default. There is no need to change these settings, unless you need to troubleshoot an issue.

Data model acceleration searches

Another problem that can be solved by the event type and tags optimization is specifically targeted at data model acceleration. With a configuration setting in the `datamodels.conf` file, you can specify a list of tags that you want to use with the data model acceleration searches. You specify the list under the `[dm.name]` stanza with the `tags_whitelist` setting.

For detailed information about the `tags_whitelist` setting, including usage examples, see Set a tag whitelist for better data model performance in the *Knowledge Manager Manual*.

Analyze search optimizations

You can analyze the impact of the built-in optimizations by using the Job Inspector.

Determine search processing time

You can use the Job Inspector to determine whether the built-in search optimizations are helping your search to complete faster.

1. Run your search.
2. From the search action buttons, select **Job** > **Inspect job**.
3. In the Job Inspector window, review the message at the top of the window. The message is similar to "The search has completed and returned X results by scanning X events in X seconds." Make note of the number of seconds to complete the search.

4. Close the Job inspector window.
5. In the Search bar, add `|noop search_optimization=false` to the end of your search. This turns off the built-in optimizations for this search.
6. Run the search.
7. Inspect the job and compare the message at the top of the Job Inspector window with the previous message. This message specifies how many seconds the search processing took without the built-in optimizations.

View the optimized search

You can compare your original search with the optimized search that is created when the built-in optimizations are turned on.

1. Run your search.
2. From the search action buttons, select **Job**, **Inspect job**.
3. In the Job Inspector window, expand the **Search job properties** section and scroll to the **normalizedSearch** property. This property shows the internal search that is created based on your original search.
4. Scroll to the **optimizedSearch** property. This property shows the search that is created, based on the **normalizedSearch**, when the built-in optimizations are applied.

Optimization settings

By default, the built-in optimizations are turned on.

Turn off optimization for a specific search

In some very limited situations, the optimization that is built into the search processor might not optimize a search correctly. In these situations, you can troubleshoot the problem by turning off the search optimization for that specific search.

Turning off the search optimization enables you to determine if the cause of unexpected or limited search results is the search optimization.

You turn off the built-in optimizations for a specific search by using the `noop` command.

You add `noop search_optimization=false` at the end of your search. For example:

```
| datamodel Authentication Successful_Authentication search | where
Authentication.user = "fred" | noop search_optimization=false
```

Turn off all optimizations

You can turn off all of the built-in optimizations for all users.

Prerequisites

- Only users with file system access, such as system administrators, can turn off the built-in optimizations.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Steps

1. Open the local `limits.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.

   Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location.

2. Under the `[search_optimization]` stanza, set `enabled` to `false`.

See also

- About optimization
- Quick tips for optimization
- Write better searches

Search normalization

When you use the `search` or `where` command in a search string, the **SPL** processor might reorder the expression statement that follows the command for normalization purposes. The SPL processor applies two kinds of normalization logic to search strings: predicate flip and predicate sort.

For more information about predicates and predicate-based search optimization, see Built-in optimization.

Use the **Job Inspector** to see the results of search normalization and optimization. See Analyze search optimizations.

Benefits of search normalization

Some search optimizations perform better after search normalization. For example, the predicate merge optimization cannot merge `where` statements that place a field value before a field name. However, if the SPL processor applies predicate flip normalization to those statements so that the field name precedes the field value, the predicate merge optimization can merge it.

See Predicate merge.

Predicate flip normalization

Under predicate flip normalization, the SPL processor takes `where` statements that have field-value pairs where field values are placed ahead of field names and switches them so that the field names come first.

For example, in this search, the field value has been placed ahead of the field:

```
index=main | where "error"=status
```

After normalization, the field name and field value are flipped:

```
index=main | where (status == "error")
```

Predicate flip normalization only works when the SPL processor can distinguish the field name from the field value. The SPL processor puts numeric field values and string field values that are surrounded by quotes on the right side of the operator. When it is possible, the SPL processor flips value-field combinations where the value includes functions, such as `value()=field`.

The SPL processor will not apply predicate flip normalization to boolean, time, and IPv4 fields. For example, with a boolean value-field pair like `true=purchased`, the SPL processor cannot distinguish whether `true` or `purchased` is the field name.

Predicate sort normalization

Under predicate sort normalization, the SPL processor uses lexicographical sorting logic to ensure that `search` expressions and `where` statements are consistently ordered in the same way.

Predicate sort for the search command

When you use the `search` command in a string, the SPL processor applies predicate sort normalization to any boolean expressions that follow it.

For example, the following three searches use the `search` command with a boolean expression. These searches look different, but they produce the same result:

```
| search ( z OR y AND d AND c AND b AND a )
```

```
| search ( d AND z OR y AND c AND b AND a )
```

```
| search ( d AND ( z OR y ) AND ( c AND b AND a ) )
```

After normalization, those strings are reordered so that they share the following form:

```
| search ((y OR z) a b c d)
```

Predicate sort for the where command

When you use the `where` command in a string, the SPL processor applies predicate sort normalization to any boolean or arithmetic statements that follow it.

For example, these `where` statements have mathematical expressions that all resolve to the same result, but are ordered differently:

```
| where x = (d+(c-a)+c*b)*b
```

```
| where b*(d+(c-a)+c*b) = x
```

```
| where ((b*c)+d+(c-a))*b = x
```

After normalization, these `where` statements share the following form:

```
| where (x == ((((b * c) + (c - a)) + d) * b))
```

Example combining predicate flip and predicate sort

The following example combines predicate flip and predicate sort. Before normalization, you can have the following `where` statements:

```
| where status="error" OR code=500
```

```
| where "error"=status OR code=500

| where 500=code OR "error"=status
```

After normalization, these `where` statements share the following form:

```
| where ((code == 500) OR (status == "error"))
```

Disable search normalization

If you put your `search` expressions and `where` statements in a specific order for search performance reasons, you might want to disable search normalization. Predicate flip normalization and predicate sort normalization are controlled by separate settings in `limits.conf`. You can disable one kind of normalization and leave the other enabled.

Prerequisites

- Only users with file system access, such as system administrators, can disable search normalization.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make changes to the files in the `local` directory.

Disable predicate flip normalization

1. Open the local `limits.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Under the `[search_optimization::search_flip_normalization]` stanza, set `enabled=false`.

Disable predicate sort normalization

1. Open the local `limits.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Under the `[search_optimization::search_sort_normalization]` stanza, set `enabled=false`.

Retrieve Events

About retrieving events

When you search, you are seeking to match search terms against segments of your event data. These search terms are keywords, phrases, boolean expressions, field name and value pairs, and so forth that specify which events you want to retrieve from the indexes. Read the Search command primer to learn how to use the search command effectively.

Your event data might be partitioned into different indexes and across distributed search peers. Read more about how to search across multiple indexes and servers in Retrieve events from indexes.

Events are retrieved from the indexes in reverse time order. The results of a search are ordered from most recent to least recent by default. You can retrieve events faster if you filter by time, whether you are using the timeline to zoom in on clusters of events or applying time ranges to the search itself. For more information, read how to Use the timeline to investigate events and About time ranges in search.

Events, event data, and fields

The phrase *event data* refers to your data after it has been added to the Splunk index. Events are a single record of activity or instance of this event data. For example, an event might be a single log entry in a log file. Because the Splunk software separates individual events by their time information, an event is distinguished from other events by a timestamp.

Here is a sample event:

```
172.26.34.223 - - [01/Jul/2005:12:05:27 -0700] "GET
/trade/app?action=logout HTTP/1.1" 200 2953
```

Events contain pairs of information, or fields. When you add data and it gets indexed, the Splunk software automatically extracts some useful fields for you, such as the host the event came from and the type of data source it is.

Use fields to retrieve events

Fields are searchable name/value pairings in **event data**. All fields have names and can be searched with those names. Searches with field expressions are more precise (and therefore more efficient) than searches using only keywords and quoted phrases.

Look at the following search:

```
host=webserver
```

In this search, `host=webserver` indicates that you are searching for events with `host` fields that have values of `webserver`. When you run this search, events with different `host` field values are not retrieved, nor are events that contain other fields that share `webserver` as a value. This means that this search returns a more focused set of results than you might get if you just searched for `webserver` in the search bar.

For more information, read "About fields" in the Knowledge Manage Manual.

Index-time and search-time fields

As Splunk software processes event data, it extracts and defines fields from that data, first at index time, and again at search time.

See "Index time versus search time" in the *Managing Indexers and Clusters* manual.

Field extraction at index time

At **index time**, Splunk software extracts a small set of fields. This set of fields includes **default fields**, custom indexed fields, and fields indexed from structured data.

Default fields exist in all events. Three important default fields are host, source, and source type. They describe where the event originated. Other default fields include datetime fields, which provide additional searchable granularity to event timestamps. Splunk software also automatically adds default fields classified as internal fields.

Custom indexed fields are fields that you have manually configured for index-time extraction. See "Create custom fields at index time" in the *Getting*

Data In manual.

Finally, when Splunk software indexes structured data, it creates index-time field extractions for the fields that it finds. Examples of structured data include:

- comma-separated value files (CSV)
- tab-separated value files (TSV)
- pipe-separated value files
- JavaScript Object Notation (JSON) data sources

When searching for default field values and custom indexed field values you can use the standard `<field>=<value>` syntax. This syntax matches default fields, custom indexed fields, and search-time fields.

However if you are searching specifically for a field that has been extracted at index-time from structured data, you can search more efficiently if you exchange the equal sign for a double colon, as follows:

```
<field>::<value>
```

This syntax works best in searches for fields that were indexed from structured data. However, you can use it to search for default and custom indexed fields as well. You cannot use it to search on **Search-time** fields.

For more information about extracting fields from structured data files, see "Extract data from files with headers" in the *Getting Data In* manual.

Field extraction at search time

At **search time**, Splunk software extracts additional fields, depending on its **Search Mode** setting and whether or not that setting enables field discovery given the type of search being run.

Search examples

Example 1: Search for events on all "corp" servers for accesses by the user "strawsky". It then reports the 20 most recent events.

```
host=corp* eventtype=access user=strawsky
```

In this example, `host` is a default field, while `eventtype` and `user` are additional fields that might have been automatically extracted or that you defined.

In general, an event type is a user-defined field that simplifies search by letting you categorize events. You can save a search as an event type and quickly retrieve those events using the `eventtype` field. For more information, read "About event types" in the Knowledge Manager Manual.

Example 2: Search for events from the source "/var/www/log/php_error.log".

```
source="/var/www/log/php_error.log"
```

The source of an event is the name of the file, stream, or other input from which the event originates.

Example 3: Search for all events that have an Apache web access source type.

```
sourcetype="access_*"
```

The source type of an event is the format of the data input from which it originates. In this search uses a wildcard to match any Apache web access log that begins with "access_". This includes access_common and access_combined (and you might also see access_combined_wcookie).

Example 4: Search indexed information from various CSV files to get a list of Plano-based employees.

```
employee_office::Plano
```

You have indexed several CSV files of employee records. Each of these CSV files share the same fields. You want to search for the employees from these files that are affiliated with the office in Plano, Texas.

This example uses the `<field>::<value>` syntax to find the fields from those CSV files, which are extracted at index time. This syntax works best for fields extracted from indexed structured data, although it can handle other kinds of index time fields as well. It cannot find fields that are extracted at search time.

Example 5: Search corp1 for events that have more than 4 lines, and omit events that contain the term 400.

```
host=corp1 linecount>4 NOT 400
```

You can use comparison expressions to match field/value pairs. Comparison expressions with "=" and "!=" work with all field/value pairs. Comparison expressions with < > <= >= work only with fields that have numeric values. This example specifies a search for events that have more than 4 lines, `linecount>4`.

Example 6: Searching with the boolean "NOT" versus the comparison operator "!=" is not the same. The following search returns events where `field` is undefined (or NULL).

```
NOT field="value"
```

The following search returns events where `field` exists and does not have the value "value".

```
field!="value"
```

In the case where the value in question is the wildcard "*", `NOT field=*` will return events where field is null/undefined, and `field!=*` will never return any events.

More about fields

This topic only discussed a handful of searches with fields.

- You can restrict searches to specific indexes and, in distributed topologies, to specific search peers.
- You can see more search examples "Using default fields" in the Knowledge Manager Manual.

Fields become more important when you start using the Splunk search language to summarize and transform your data into reports. For more information, read "About reporting commands".

Event sampling

By default, a Splunk search retrieves all events. However in some situations you might want to retrieve a sample set of events, instead of retrieving the entire event set. There are several reasons why you might want to use event sampling.

◊ To perform a quick search to ensure the correct events are being returned
◊ To determine the characteristics of a large data set without processing every event
◊ To test that the data selection, formatting, calculations, and other components of the search are working correctly

For most searches, event sampling can greatly increase search performance without decreasing functionality.

The event sampling ratio

The sampling ratio is the likelihood of any event being included in the sample result set. The formula for the ratio is `1/sample_ratio_value`.

For example, if the sample ratio value is 100, each event has a 1 in 100 chance of being included in the result set. The selection of each event is independent of the selection of all another events. It is possible that many events are included from the first 100 events, or none at all.

If a search matches 1,000,000 events when sampling is not used, using a sample ratio value of 100 would result in returning approximately 10,000 events.

If you to rerun a sampling search many times, the exact number of returned results is modeled by a binomial distribution with n=1000000 and p=0.01. This distribution looks like a normal distribution, with the mean=10000 and the standard deviation (stdev)=99.5.

In Splunk Web, the sampling ratio that you specify must be a positive integer that is greater than 1. To disable sampling in Splunk Web, set the ratio to 1.

Set the default sampling ratio

In Splunk Enterprise, set the default sampling ratio by editing the `ui-prefs.conf` file. The sampling ratio must be a positive integer.

In Splunk Cloud, to change the default sampling ratio, file a Support ticket.

How event sampling works

By default, event sampling is not active. When you run a search, every event that matches your criteria is returned. When you specify a ratio, sampling remains in effect for the active search window. Sampling also remains in effect when you save a search as a report or dashboard panel.

When you specify a ratio value, your value overrides the default value configured for your Splunk deployment and remains in effect until you change it.

If you open a new search window, event sampling is no longer active. However, the last custom ratio that you used appears in the **Sampling** drop-down.

Commands and functions to avoid with event sampling

Typically, searches that use the `transaction`, `stats`, or `streamstats` commands are not good candidates for sampling.

When you calculate statistics using a sample set of events, the statistical values will not be accurate. To determine the true statistical value, you must scale the value returned with event sampling. And scaling only gives you an approximate true value.

For example, you create a report using this search with event sampling enabled.

```
... | stats sum(x)
```

Because you used event sampling, the returned value is not the complete sum of all of the events. It is only the sum of the sample set of events. If the sampling ratio is 100, the true sum is approximately 100 times the value returned by the search.

Statistical calculations that fall into this situation are `count`, `sum`, and `sumsq`.

Other statistics that are difficult to interpret when event sampling is used include:

- `distinct_count`
- `earliest`
- `latest`
- `max`
- `min`

Specify a sampling ratio

You activate event sampling for a search by specifying a sampling ratio.

1. In Splunk Web, below the Search bar, click **No Event Sampling**.

2. You can use one of the default ratios or specify a custom ratio.

a. To use one of the default ratios, click the ratio in the **Sampling** drop-down.
b. To specify a custom ratio, click **Custom** and type the ratio value. Then click **Apply**. The ratio value must be a positive integer greater than 1.

Event sampling indicators

There are several indicators in the Search & Reporting App window which show that event sampling is active. After you run a search, the **Sampling** drop-down appears in the event count line. The label for the **Sampling** drop-down specifies the ratio that is applied to the search. Additionally, if a sampling ratio is being used, the **Jobs** drop-down specifies the ratio that is applied to the search.

Event sampling with reports and dashboard panels

You can save a search that uses event sampling as a report or dashboard panel. Use the **Save As** drop-down to save the search.

When the search is saved as a report, the sampling ratio is used when the report is run.

When the search saved as a dashboard panel, the panel is powered by an inline search. When the dashboard is refreshed, the sampling ratio that was saved with the inline search is used.

If you open a report and add the report to a dashboard panel, you can specify how the panel is powered. You can specify that the panel is powered by the inline search that the report is based on. Or you can specify that the panel is powered by the report itself.

Panels powered by reports
: When you view the source for the panel in Simple XML, there is no indication if the report uses event sampling.

Panels powered by inline searches
: When you view the source for the panel in Simple XML, if the underlying search uses event sampling there is <sampleRatio> entry. For example:

```
<event>
  <title>sample events</title>
  <search>
     <query>buttercupgames</query>
     <earliest>@d</earliest>
     <latest>now</latest>
     <sampleRatio>500</sampleRatio>
  </search>
</event>
```

Accelerated reports

You cannot accelerate reports that are based on event sampling searches. See "Accelerate reports" in the *Reporting Manual*.

Retrieve events from indexes

You have always been able to create new indexes and manage where you want to store your data. Additionally, when you have data split across different indexes, you can search multiple indexes at once, using the `index` field.

Specify one or multiple indexes to search

The Splunk administrator can set the default indexes that a user searches. Based on the roles and permissions, the user might have access to one or many indexes. For example the user might be able to only search main or all public indexes. The user can then specify a subset of these indexes, either an individual index or multiple indexes, to search. For more information about setting up users and roles, see "About users and roles" in *Securing Splunk Enterprise*.

For more information about managing your indexes and setting up multiple indexes, see the "About managing indexes" in the *Managing Indexers and Clusters* manual.

Control index access using Splunk Web

1. Navigate to **Manager > Access controls > Roles**.

2. Select the role that the User has been assigned to.

On the bottom of the next screen you'll find the index controls.

3. Control the indexes that particular role has access to, as well as the default search indexes.

Syntax

You can specify different indexes to search in the same way that you specify field names and values. In this case, the field name is `index` and the field value is the name of a particular index:

```
index=<indexname>
```

You can use the * wildcard to specify groups of indexes; for example, if you wanted to search both "mail" and "main" indexes, you can search for:

```
index=mai*
```

You can also use parentheses to partition different searches to certain indexes. See Example 3 for details.

Note: When you type "index=" into the search bar, typeahead indicates all the indexes that you can search, based on your roles and permissions settings.

Examples

Example 1: Search across all public indexes.

```
index=*
```

Example 2: Search across all indexes, public and internal.

```
index=* OR index=_*
```

Example 3: Partition different searches to different indexes; in this example, you're searching three different indexes: main, _internal, and mail. You want to see events that match "error" in all three indexes; but also, errors that match "warn" in main or "failed" in mail.

```
(index=main (error OR warn)) OR (index=_internal error) OR (index=mail
(error OR failed))
```

Example 4: Search across multiple indexes on different distributed Splunk servers.

```
(splunk_server=local index=main 404 ip=10.0.0.0/16) OR
(splunk_server=remote index=mail user=admin)
```

Not finding the events you're looking for?

When you add an input, the input gets added relative to the app you're in. Some apps write input data to their own specific index (for example, the Splunk App for Unix and Linux uses the 'os' index).

Search across one or more distributed search peers

When performing a distributed search from a search head, you can restrict your searches to specific search peers (also known as "indexer nodes") by default and in your saved and scheduled searches. The names of your Splunk search peers are saved as values in the "splunk_server" field. For more information about distributed search, see "About distributed search" in the Distributed Search manual.

If no search peer is specified, your search accesses all search peers you have permission to access. The default peers that you can access are controlled by the roles and permissions associated with your profile and set by your Splunk admin. For more information, see "About users and roles" in Securing Splunk Enterprise.

The ability to restrict your searches to specific peers can be useful when there is high latency to certain search peers and you do not want to search them by default. When you specify one or more peers, those are the only servers that are included in the search.

You can specify different peers to search in the same way that you specify other field names and values. In this case, the field name is "splunk_server" and the field value is the name of a particular distributed peer:

```
splunk_server=<peer_name>
```

Note: You can use the value "local" to refer to the Splunk instance that you are searching from; in other words, the search head itself.

```
splunk_server=local
```

Keep in mind that **field names are case sensitive**; Splunk will not recognize a field name if the case doesn't match.

Examples

Example 1: Return results from specified search peers.

```
error (splunk_server=NYsplunk OR splunk_server=CAsplunk) NOT
splunk_server=TXsplunk
```

Example 2: Search different indexes on distributed search peers "foo" or "bar".

```
(splunk_server=foo index=main 404 ip=10.0.0.0/16) OR (splunk_server=bar
index=mail user=admin)
```

Classify and group similar events

An event is not the same thing as an event type. An event is a single instance of data — a single log entry, for example. An event type is a classification used to label and group events.

The names of the matching event types for an event are set on the event, in a multivalue field called `eventtype`. You can search for these groups of events (for example, SSH logins) the same way you search for any field value.

This topic discusses how to classify events (save a search as an event type) and search for tagged fields. For more information about events, how Splunk software recognizes them, and what it does when it processes them for indexing, see the Overview of event processing topic in the *Getting Data* In manual.

Important: You cannot save a search pipeline as an event type; that is, when saving a search as an event type, it cannot include a search command.

Save a search as a new event type

When you search your event data, you are essentially filtering out all unwanted events. The results of your search are events that share common characteristics, and you can give them a collective name.

For example, if you often search for failed logins on different host machines, you can save an event type for the events and call it `failed_login`:

```
"failed login" OR "FAILED LOGIN" OR "Authentication failure" OR "Failed
to authenticate user"
```

To save this search as an event type:

1. Click **Save As** and select **Event Type**.
2. In **Save As Event Type** window, give your event type a **Name**.
   In this example the name is **failed_login**.

You can add a list of tags that should be applied to the event type in the **Tag(s)** field. For more about tags see the section **Use tags to group and find similar events** below.

3. Click **Save** to save your event type name.

Now, you can quickly search for all the events that match this event type the same way you can search for any field, by specifying the event type in your search criteria.

For example, you might be interested in finding failed logins on specific host machines. Your search might look something like this:

```
host=target eventtype=failed_login
```

Or you might want to investigate suspicious user activity. Your search might look something like this:

```
user=suspicious eventtype=failed_login
```

Use typelearner to discover new event types

Pass any of your searches into the `typelearner` command to display suggestions for event types. By default, `typelearner` compares the punctuation of the events resulting from the search, grouping those that have similar punctuation and terms together.

You can specify a different field for Splunk software to group the events; `typelearner` works the same way with any field. The result is a set of events (from your search results) that have this field and phrases in common.

For more information and examples, see "typelearner" in the search command reference.

Use tags to group and find similar events

In your data, you might have groups of events with related field values. To help you search more efficiently for these groups of fields, you can assign tags to their field values. You can assign one or more tags to any extracted field (including event type, host, source, or source type).

Event types can have one or more tags associated with them. You can add these tags while you save a search as an event type and from the event type manager, located in **Manager > Event types**. From the list of event types in this window, select the one you want to edit.

After you add tags to your event types, you can search for them in the same way you search for any tag. Let's say you saved a search for firewall events as the event type `firewall_allowed`, and then saved a search for login events as the event type `login_successful`. If you tagged both of these event types with *allow*, all events of either of those event types can be retrieved by using the search:

```
tag::eventtype="allow"
```

You can tag field/value pairs. You can also alias field names. See Tag field value pairs in Search and Create field aliases in Splunk Web.

Search for tagged field values

There are two ways to search for tags. If you are searching for a tag associated with a value on any field, you can use the following syntax:

```
tag=<tagname>
```

Or, if you are looking for a tag associated with a value on a specific field, you can use the following syntax:

```
tag::<field>=<tagname>
```

Use wildcards to search for tags

You can use the asterisk (`*`) wildcard when searching keywords and field values, including for eventtypes and tags.

For example, if you have multiple event-type tags for various types of IP addresses, such as `IP-src` and `IP-dst`, you can search for all of them with:

```
tag::eventtype=IP-*
```

If you wanted to find all hosts whose tags contain "local", you can search for the tag:

```
tag::host=*local*
```

Also, if you wanted to search for the events with eventtypes that have no tags, you can search for the Boolean expression:

```
NOT tag::eventtype=*
```

Use the timeline to investigate events

The timeline is a visual representation of the number of events in your search results that occur at each point in time. The timeline shows the distribution of events over time.

When you use the timeline to investigate events, you are not running a new search. You are filtering the existing search results.

You can use the timeline to highlight patterns or clusters of events or investigate peaks (spikes in activity) and lows (possible server downtime) in event activity. Position your mouse over a bar to see the count of events. Click on a bar to drill-down to that time range.

Change the timeline format

The timeline is located in the Events tab above the events listing. It shows the count of events over the time range that the search was run. Here, the timeline shows web access events over **All time**.

Format options are located in the **Format Timeline** menu:

You can hide the timeline, or display a Compact or Full view of the timeline. You can also toggle the timeline scale between Linear scale or Log scale (logarithmic).

When **Full** is selected, the timeline view is taller to accommodate the labels on the axis. The count is on the Y-axis and time is on the X-axis.

Zoom in and zoom out to investigate events

Above the timeline are the zoom options. By default, the timeline is zoomed in. The following image shows the timeline display in Full view and zoomed in. The **Zoom Out** option is available.

Timeline legend

The timeline legend is on the top right corner of the timeline. This indicates the scale of the timeline. For example, **1 hour per column** indicates that each column represents a count of events during that hour. Zooming in and out changes the time scale.

Zoom out

When you click **Zoom Out**, the legend indicates that each column now represents **1 day per column** instead of an hour.

Zooming out changes not only the timeline but the value in the Time Range Picker.

Reset the zoom

To reset the zoom or to zoom in, change the value in the Time Range Picker. For example, if you searched using **All time** and then zoomed out, select **All time** in the Time Range Picker to return to the original timeline time scale.

Zoom to a selection

When you mouse over and select bars in the timeline, the **Zoom to Selection** or **Deselect** options above the timeline become available.

Mouse over and click on the tallest bar or drag your mouse over a cluster of bars in the timeline. The events list updates to display only the events that occurred in that selected time range. The time range picker also updates to the selected time range. You can cancel this selection by clicking **Deselect**.

When you select a set of bars on the timeline and click **Zoom to Selection**, your search results are filtered to show only the selected time period. The timeline and events list update to show the results of your selection.

The dates and times that correspond to the bars you selected, along with the number of events in that time range, is reflected in the information just below the Search bar.

You cannot **Deselect** after you zoomed into a selected time range. But, you can **Zoom Out** again or change the time in the Time Range Picker.

Drill down on event details

After running a search that returns events in the **Events** tab, click on parts of those events to run different kinds of secondary **drilldown** searches that use the event detail that you have selected.

In the **Events** tab you can run a drilldown search when you click on these parts of an event:

- **Field value**. Fields are **extracted** from events at index time and search time. See "About fields" in the *Knowledge Manager Manual*.
- **Tag**. A **tag** is associated with one or more field-value pairs. A field-value pair can be associated with multiple tags. See About tags and aliases in the *Knowledge Manager Manual*.
- **Segment** (can be a connected string of segments). A **segment** is a searchable part of an event. See "About event segmentation" in the *Getting Data In Manual* to learn how segments are configured and created.
- **Timestamp**. A **timestamp** is date and time information in an event.

Drilldown search actions

The drilldown searches can perform the following actions for fields, tags, and event segments.

Drilldown search action	Description	Result
Add to search	Add a focus on the selected event detail to the original search string and run it. Splunk software discards **transforming commands** from the search string, as well as anything that follows them.	A dataset similar to the one returned by the original search, filtered to include only events

		that have the selected field, tag, or segment(s).
Exclude from search	Add an exclusion of the selected event detail to the original search and run it. Splunk software discards transforming commands from the search string, as well as anything that follows them.	A dataset similar to the one from the original search, filtered to include only events that do not have the selected field, tag, or segment(s).
Remove from search	Run a new search that is identical to the original search, except that it no longer searches for the highlighted field value, tag, or segment. Splunk software discards transforming commands from the search string, as well as anything that follows them.	A new dataset that is not as restrictive as the dataset returned by the original search. It no longer excludes events without the highlighted field value, tag, or segment.
New search	Run a new search that focuses exclusively on the selected field, tag, or segment(s).	A new dataset that contains any event which includes the selected field, tag, or segment(s).

Examples of drilldown search actions

These examples use the sample data from the Search Tutorial but should work with any format of Apache web access log. To try these examples on your own Splunk instance, you must download the sample data and follow the instructions to **get the tutorial data into Splunk**. Use the time range **Last 7 days** when you run the searches.

Add to search

1. Run a search or report that returns an event listing in the **Events** tab. If your search includes **transforming commands**, set the **Search Mode** to **Verbose**.

   Let's start with a basic search that searches the Apache web access log files and looks for HTTP errors that start with 4, such as 404.

```
sourcetype=access_* status=4*
```

2. In the search results, expand the event information for the first event.
3. Select the `clientip` field value. In this example the value is `182.236.164.11`.

   Because the `clientip` field is not currently in the search, the options are: **Add to search**, **Exclude from search**, and **New search**.

   Field values, tags, and segments that are explicitly included in the search string appear with yellow highlighting in the event information.
4. Click **Add to search**. The field-value pair is added to the search. The search is run using the same time range. For example:

   ```
   sourcetype=access_* status=4* clientip="182.236.164.11"
   ```

Remove from search

You can use the **Remove from search** drilldown option to remove specific criteria. You cannot remove criteria that includes a wildcard.

Continuing with the same search, let's remove the `clientip` from the search.

1. To remove the criteria from the search, expand the event information for the first search result.
2. Select the value for the `clientip` field. The drilldown options are **Remove from search** and **New search**.

3. Click **Remove from search**. The search is updated and runs using the same time range. For example:

   ```
   sourcetype=access_* status=4*
   ```

Suppose the original search looks like this:

```
sourcetype=access_* status=404
```

Expand any event information that contains the `status` criteria. Click on the value `404` for the status field. From the drilldown options, select **Remove from search**.

Exclude from search

You can use the **Exclude from search** drilldown option to exclude a specific criteria from the search.

1. Expand the event information for a search result that contains the criteria that you want to exclude.
   Suppose the original search look like this:

   ```
   sourcetype=access_* status=4*
   ```

   You want to exclude all **view** actions.
2. For the `action` field, select the **view** value and click **Exclude from search**. The field-value pair is added to the search with the **not equal to** syntax. The search is run using the same time range. For example:

   ```
   sourcetype=access_* status=4* action!=view
   ```

Selecting segments

To select event segments or sets of connected segments before you click on them. Splunk software identifies your segment selection with yellow highlighting.

Common values

If the value you select is among the top ten values found for the field, the **Add to search** and **Exclude from search** options show event numbers below the option names.

- The event number under **Add to search** indicates how many events contain the value for the field.
- The event number under **Exclude from search** indicates how many events do not contain the value for the field.

Run the drilldown search in a new tab

When you select one of the drilldown options, the search is run in the current tab and replaces your current search.

However, you can run the drilldown search in a new tab and leave your current search results intact. To run the drilldown search in a new tab, click the **Open In New Tab** icon for the option.

Drilldown searches for event timestamps

Event timestamp drilldown searches can help you find event correlations and perform root cause analysis.

Click on an event **timestamp** to run a secondary search that returns events which are chronologically close to that event.

When you open an event you can also click on the `_time` field to run this kind of drilldown search.

The controls for this search are called a **_time accelerator**. See Use time to find nearby events in this manual for details on how the _time accelerator is used.

Identify event patterns with the Patterns tab

Events in search results can be grouped into **event patterns**. Events that belong to an event pattern share common characteristics, and usually can be returned by a specific search string. Event pattern analysis is useful for searches that return a diverse range of events because it quickly shows you the most common kinds of events in your search result dataset.

The Patterns tab simplifies event pattern identification. Click the Pattern tab to view a list of the most common patterns among the set of events returned by your search. Each of these patterns represents a set of events that share a similar structure.

Click on a pattern to:

- View the approximate number of events in your results that fit the pattern.
- View the search that returns events with this pattern.
- Save the pattern search as an event type, if possible. Not all event patterns can be saved as event types.
- Create an alert based on the pattern. For example, you can create alerts that trigger when certain patterns increase or decrease in frequency.

An event patterns example

A search that uses `sourcetype=cisco:esa` runs for All time and returns 112,421 events.

The patterns are based on a sample of 50,000 events.

To view all of the patterns in the list of events, click the **Patterns** tab. Initially the results identify 45 patterns. You can move the Smaller to Larger slider to change how narrow or broad the event patterns should be. If you drag the slider closer to the **Larger** side, 14 patterns are returned.

The threat level event pattern is the most common one. Some of the listed patterns are relatively rare in this dataset, and finding them in the Events tab listing might be difficult. The Patterns tab makes it easier to see these event patterns and save them as event types if necessary.

How the Patterns tab works

When you click the Patterns tab, Splunk software runs a secondary search on a subset of the search results that have been received up to that point. This search job analyzes those results and derives the most common event patterns in those results. It then lists those patterns in descending order from most prevalent to least prevalent. It may not include outlier patterns that are based on extremely small groups of events, because they are statistically unreliable.

The secondary search can take a long time to complete when the original dataset contains an extremely large variety of event patterns. For example, some searches return datasets containing over 500 patterns, where most of those patterns represent very small collections of events. The algorithm that identifies these patterns is designed to avoid doing too much work for small patterns, but it also attempts to be as accurate as possible.

The Patterns tab only accepts **transforming searches** when you set their **search mode** to **Verbose**. The Patterns tab cannot find patterns for **real-time searches** in any search mode.

Event pattern keywords

The Patterns tab defines patterns by the presence or absence of one or more keywords. If the keywords identified for a pattern are added to or excluded from the original search, the search returns events that fit that pattern. Keywords present in a pattern are identified with green text in the pattern list. Excluded keywords are not identified in the event list.

In the preceding event pattern example, the threat level event pattern has "threat" as its keyword, meaning that the search that returns events fitting the pattern would look like this:

```
sourcetype=cisco_esa threat
```

If this event pattern were also identified by the exclusion of the keyword "verified," the search that returns events fitting the pattern would look like this:

```
sourcetype=cisco_esa threat ( NOT verified )
```

To see all of the keywords associated with a pattern, click on the pattern.

Use the Patterns tab

1. From the Search view, run a search that returns more than 5000 results.

    Searches returning more than 5000 results produce reliable patterns.

2. Click the Patterns tab.

    You do not need to wait for the search to complete, but the pattern listing is more accurate with finalized search results.

3. (Optional) If there appear to be too many patterns or too few, or if you do not see the patterns you expect, move the slider.

    Dragging the slider to **Larger** runs a secondary search job that consolidates some patterns together, resulting in event patterns that represent larger numbers of events, and a wider variety of events inside each pattern group.

    Dragging the slider to **Smaller** runs a secondary search job that increases the granularity of the results. The event patterns it finds represent smaller numbers of events.

4. (Optional) Click on a pattern to view information for that pattern.

Estimated Events is the estimated count of events in the dataset returned by the original search that fit the event pattern. In this example, the original search had 112k events. This pattern accounts for an estimated 12,200 events or 10.84% of the total number of events.

Included Keywords identifies keywords that should be added to the base search to return the pattern. If the Patterns tab identifies keywords that should be excluded from the base search, they appear under an **Excluded Keywords** section.

You can see the search that returns events fitting the event pattern under **Search**.

5. (Optional) In the pattern information area, click **View Events** to run the search displayed under **Search**.

When it runs, this search uses the same time range as your original search.

6. (Optional) In the pattern information area, click **Save as event type** to save the search as an **event type**.

Save as event type is available only for event patterns based on searches that do not include pipe characters and additional search commands. See "About event types" in this manual.

7. (Optional) In the pattern information area, click **Create alert** to create an **alert** based on the pattern.

For example, create a scheduled alert that is triggered when the frequency of the event pattern rises above or drops below a threshold. If you know that events that fit an event pattern tend to appear at a steady rate of approximately 100 events per hour, set the alert to run on an hourly

schedule and trigger when 150 or more events are returned. See the *Alerting Manual*.

Numbers in the Patterns tab

When the secondary search finishes, the Patterns tab displays a message explaining how many events it analyzed to obtain the displayed results.

The Patterns tab analyzes a subset of the total number of events returned by the original search. The maximum number of events in this subset is 50k. This maximum reduces processing times for the Pattern tab secondary search. If your original search returns less than 50k events, the secondary search analyzes up to 1000 events per timeline bar spanned by the original search. For example, if the original search spans 14 timeline bars, the secondary search analyzes 14,000 events to obtain its patterns listing.

You can control the maximum number of events analyzed by the secondary search by updating the `maxevents` setting, in the [findkeywords] stanza, in `limits.conf`. This setting defaults to `50000`. Do not change this value. A number less than 50,000 reduces the accuracy of your events. A number higher than 50,000 increases the processing time required by the secondary search.

The estimated event count provided in the pattern information area does not apply to the number of events analyzed in the Patterns tab secondary search. It applies to the total number of events returned by the original search. If an event pattern is estimated to represent 7,350 events and the original search returned 265k events, the pattern accounts for 2.7% of the events returned by the search.

Restricting Patterns tab usage

All roles, including the user role, have permission to use the Patterns tab by default. To restrict usage of the Patterns tab, remove the `pattern_detect` **capability**. Roles without this capability do not see the Patterns tab option after they run a search.

For more information about capabilities, see "About defining roles with capabilities" in the *Securing Splunk* manual.

Preview events

In a distributed environment, by default, when you run a search the results are not displayed until all of the search peers begin to return event data for the time range that you specify. In a distributed environment with a large number of peers, or where some of the peers are slow, there can be a delay in displaying search results.

The events preview mode displays an event as soon as the event is returned, instead of waiting until all of the events are returned to see the search results. This mode displays events that are in-memory and not yet committed.

Limitations using the preview mode

There are some limitations in the Events viewer when you enable the events preview mode.

You cannot expand the Events viewer to see detailed information about an event until all of the events from your search are returned. When you position your mouse over the information icon, a message informs you that the events preview mode is enabled.

As results are returned and displayed in the Events viewer, the order of the events changes. As new results are added to the Events viewer, the events are inserted into the correct time order.

The Events viewer provides the option to display events in a list, as a table, or as raw events. When the Events viewer is set to **Table** and the events preview mode is enabled, you cannot sort the list of events until the search completes.

Enable events preview mode

To see the events more quickly, you can enable the events preview mode in the Search app.

When you enable events preview, it is enabled for everyone using the Search app, not just for you.

Prerequisites

- Only users with file system access, such as system administrators, can enable the events preview mode.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

Steps

1. Open the local `limits.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Under the [search] stanza, set `timeline_events_preview` to `true`.

If you are using Splunk Cloud and want to enable the preview mode, open a Support ticket.

Specify Time Ranges

About searching with time

How timestamps are used

Timestamp processing is a key step in **event processing**. Splunk software uses **timestamps** to:

- Correlate **events** by time
- Create the timeline histogram in Splunk Web
- Set time ranges for searches

Splunk software adds timestamps to events at **index time**. Timestamp values are assigned automatically by using information that the software finds in the raw event data. See How timestamp assignment works in *Getting Data In*.

Timestamp formats

Splunk software sometimes represents time in UNIX time. Time represented this way appears as a series of numbers, for example `1518632124`. You can use any UNIX time converter to convert the UNIX time to either GMT or your local time.

The _time field

The `_time` field is in UNIX time. In Splunk Web, the `_time` field appears in a human readable format in the UI but is stored in UNIX time.

Specify narrow time ranges

When you start a new search, the default time range is **Last 24 hours**. This range helps to avoid running searches with overly-broad time ranges that waste system resources and produce more results than you really need.

Whether you are running a new search, a report, or creating a dashboard, it is important to narrow the time range to only the dates or times that you really need.

Time is also crucial for determining what went wrong. You often know when something happened, if not exactly what happened. By looking at events that

happened around the same time that something went wrong, can help correlate results and find the root cause of the problem.

This section discusses how to use time to narrow your search and how to group events in your search by time.

- Select time ranges to apply to your search
- Specify time modifiers in your search criteria
- Specify time windows in your real-time searches
- Use time to find nearby events

Select time ranges to apply to your search

Use the **time range picker** to set time boundaries on your searches. You can restrict a search with preset time ranges, create custom time ranges, specify time ranges based on date or date and time, or work with advanced features in the time range picker. These options are described in the following sections.

Note: If you are located in a different timezone, time-based searches use the timestamp of the event from the Splunk instance that indexed the data.

Select from a list of Preset time ranges

The time range picker includes many built-in time ranges options that are already defined in the `times.conf` file. You can select from a list of Real-time windows, Relative time ranges, and search over All Time.

Define custom Relative time ranges

Use Relative time range options to specify a custom time range for your search that is relative to Now or the Beginning of the current hour. You can select from the list of time range units: Seconds Ago, Minutes Ago, and so on.

By default, Earliest is set to **No Snap-to** and Latest is set to **Now**. If you specify the snap-to option for **Earliest** or **Latest**, the time range will snap to beginning of the time frame that you select. For example, if you select **Days Ago**, the Earliest snap to value is **Beginning of today**.

The preview boxes below the fields update to the time range as you make the selections.

To learn more about relative time ranges, see Specify time modifiers in your search.

Define custom Real-time time ranges

The Real-time option enables you to specify a custom Earliest time for a **real-time search**. Because this time range is for a real-time search, a Latest time is not relevant.

To learn more about time ranges for real-time searches, see Specify real-time time range windows in your search.

Define custom Date ranges

Use the Date Range option to specify custom calendar dates in your search. You can choose among options to return events: **Between** a beginning and end date, **Before** a date, and **Since** a date.

For these fields, you can type the date into the text box or select the date from a calendar.

Define custom Date & Time ranges

Use the Date & Time Range option to specify custom calendar dates and times for the beginning and ending of your search.

You can type the date into the text box or select the date from a calendar.

Use Advanced time range options

Use the Advanced option to specify the earliest and latest search times. You can write the times in UNIX time or relative time notation, such as `-3d@d`. The UNIX time value you type is converted to local time.

The UNIX time or relative time that you specify is displayed as a timestamp under the text field so that you can verify your entry.

Customize the list of Preset time ranges

You can customize the set of time ranges that appear in the **Presets** list the time range picker in Splunk Web. You can create a time range based on an existing time range, or you can hide time ranges.

Create a time range based on an existing time range

The easiest way to create a new time range is to use an existing time range as the basis for a new time range. For example, the Relative time range list contains the **Last 15 minutes** time range. You want to create a time range for the last 30 minutes. You start by creating a duplicate, or clone, of the **Last 15 minutes** time range. In the clone, you change the **Earliest** setting from -15min to -30min.

1. From the **Settings** menu, under the Knowledge list select **User interface**.
2. In the User Interface window, select **Time ranges**.
3. Locate the time range that you want to use.
4. In the Actions column click **Clone**.
5. A copy of the specifications for the time range appear. Make the changes to the time range specifications and click **Save**.

The new time range appears in the Relative list in the Presets menu.

Create a new Preset time range

You can create a new time range for the Presets menu. For example, you want to create a time range that shows searches yesterday from the hours of 12:00 to 15:00. You need to specify relative times in the Earliest and Latest fields. In the **Earliest** field you specify `-1d@d+12h`. In the **Latest** field you specify `-1d@d+15h`.

1. From the **Settings** menu, under the Knowledge list select **User interface**.
2. In the User Interface window, select **Time ranges**.

3. Click **New**.
4. Complete the fields in the Add New window and click **Save**.

The new time range appears in the Relative list in the Presets menu.

Hide a time range on the Presets list

1. From the **Settings** menu, under the Knowledge list select **User interface**.
2. In the User Interface window, select **Time ranges**.
3. Locate the time range you want to hide. In the Status column click **Disable**.

Setting default time ranges for the API or CLI

You can set time ranges manually in the `times.conf` file when you want to specify a time range for a REST API endpoint or for the command line interface (CLI).

Prerequisites

- Only users with file system access, such as system administrators, can change time ranges manually in the `times.conf` file.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

Steps

1. Open the local `times.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Create a stanza for the time range that you want to specify. For examples, see the times.conf reference in the *Admin Manual*.

If you are using Splunk Cloud and want to either hide a time range or create a new time range, open a Support ticket.

Change the default time range

The default time range for ad hoc searches in the Search & Reporting App is set to **Last 24 hours**. An administrator can set the default time range globally, across all apps. See Change default values in the *Admin Manual*.

Specify time modifiers in your search

When searching or saving a search, you can specify absolute and relative time ranges using the following time modifiers:

```
earliest=<time_modifier>
latest=<time_modifier>
```

An absolute time range uses specific dates and times, for example, from 12 A.M. November 1, 2017 to 12 A.M. November 13, 2017.

A relative time range is dependent on when the search is run. For example, a relative time range of -60m means 60 minutes ago. If the current time is 3 P.M., the search returns events from the last 60 minutes, or 2 P.M. to 3 P.M. today.

The current time is referred to as **Now**.

Time modifiers and the Time Range Picker

A time range that you specify in the Search bar, or in a saved search, overrides the time range that is selected in the Time Range Picker.

Time ranges and subsearches

Time ranges that you specify directly in a search apply only to the main search. The time ranges specified in the main search do not apply to subsearches. Likewise, a time range specified in a subsearch applies only to that subsearch. The time range does not apply to the main search or any other subsearch.

Time ranges selected from the Time Range Picker apply to the main search and to subsearches.

Specify absolute time ranges

For exact time ranges, the syntax for the time modifiers is `%m/%d/%Y:%H:%M:%S`. For example, the following search specifies a time range from 12 A.M. October 19, 2017 to 12 A.M. October 27, 2017.

```
earliest=10/19/2017:0:0:0 latest=10/27/2017:0:0:0
```

If you specify only the `earliest` time modifier, `latest` is set to the current time **Now** by default. If you specify a `latest` time modifier, you must also specify an `earliest` time.

Specify relative time ranges

You define the relative time in your search by using a string of characters that indicate the amount of time. The syntax is an integer and a time unit.

1. Begin your string with a minus (-) or a plus (+) to indicate the offset before or after the time amount.

2. Specify the amount of time by using a number and a time unit. When you specify single time amounts, the number is implied. For example `s` is the same as `1s`, `m` is the same as `1m`, and so on. The supported time units are listed in the following table.

Time range	Valid values
seconds	s, sec, secs, second, seconds
minutes	m, min, minute, minutes
hours	h, hr, hrs, hour, hours
days	d, day, days
weeks	w, week, weeks
months	mon, month, months
quarters	q, qtr, qtrs, quarter, quarters
years	y, yr, yrs, year, years

When specifying relative time, use `Now` to refer to the current time.

Relative time modifiers that snap to a time

With relative time, you can specify a **snap to** time, which is an offset from the relative time. The snap to time unit rounds down to the nearest or latest time for the time amount that you specify. To do this, separate the time amount from the snap to time unit with an "@" character.

The syntax for the snap to time unit is `[+|-]<time_integer><time_unit>@<time_unit>`.

When snapping to the nearest or latest time, Splunk software always snaps backwards or rounds down to the latest time that is not after the specified time. For example, the current time is 15:45:00 and the snap to time is `earliest=-h@h`. The time modifier snaps to 14:00.

You can also define the relative time modifier using only the snap to time unit. For example, to snap to a specific day of the week, use @w0 for Sunday, @w1 for Monday, and so forth. For Sunday, you can specify either w0 or w7.

If you do not specify a snap to time unit, the search uses seconds as the snap to time unit.

The snap to option becomes very useful in a range of situations. For example, if you want to search for events in the previous month, specify `earliest=-mon@mon latest=@mon`. This example begins at the start of the previous month and ends at the start of the current month.

Difference between relative time and relative snap to time

On April 28th, you decide to run a search at 14:05.

- If you specify `earliest=-2d`, the search goes back exactly two days, starting at 14:05 on April 26th.
- If you specify `earliest=-2d@d`, the search goes back to two days and snaps to the beginning of the day. The search looks for events starting from 00:00 on April 26th.

Special time units

The following abbreviations are reserved for special cases of time units and snap time offsets.

Time Unit	Description
`earliest=1`	If you want to search events from the start of UNIX epoch time, use `earliest=1`. UNIX epoch time 1 is UTC January 1, 1970 at 12:00:01 AM.

	`earliest=0` in the search string indicates that time is not used in the search. When `earliest=1` and `latest=now` or `latest=<a large number>`, the search will run over **all time**. The difference is that: • Specifying `latest=now()` (which is the default) does not return future events. • Specifying `latest=<a big number>` returns future events, which are events that contain timestamps later than the current time, `now()`.
`latest=now()`	Specify that the search starts or ends at the current time.
`@q, @qtr, or @quarter`	Specify a snap to the beginning of the most recent quarter: Jan 1, Apr 1, July 1, or Oct 1.
`w0, w1, w2, w3, w4, w5, w6, and w7`	Specify "snap to" **days of the week**; where w0 is Sunday, w1 is Monday, etc. When you snap to a week, `@w` or `@week`, it is equivalent to snapping to Sunday or `@w0`. You can use either w0 or w7 for Sunday.

Examples of relative time modifiers

For these examples, the current time is Wednesday, 05 February 2017, 01:37:05 P.M. Also note that 24h is usually but not always equivalent to 1d because of Daylight Savings Time boundaries.

Time modifier	Description	Resulting time	Equivalent modifiers
now	Now, the current time	Wednesday, 05 February 2017, 01:37:05 P.M.	now
-60m	60 minutes ago	Wednesday, 05 February 2017, 12:37:05 P.M.	-60m@s
-1h@h	1 hour ago, to the hour	Wednesday, 05 February 2017, 12:00:00 P.M.	
-1d@d	Yesterday	Tuesday, 04 February 2017, 12:00:00 A.M.	
-24h	24 hours ago (yesterday)	Tuesday, 04 February 2017, 01:37:05 P.M.	-24h@s

-7d@d	7 days ago, 1 week ago today	Wednesday, 28 January 2017, 12:00:00 A.M.	
-7d@m	7 days ago, snap to minute boundary	Wednesday, 28 January 2017, 01:37:00 P.M.	
@w0	Beginning of the current week	Sunday, 02 February 2017, 12:00:00 A.M.	
+1d@d	Tomorrow	Thursday, 06 February 2017, 12:00:00 A.M.	
+24h	24 hours from now, tomorrow	Thursday, 06 February 2017, 01:37:05 P.M.	+24h@s

Examples of chained relative time offsets

You can also specify offsets from the snap-to-time or "chain" together the time modifiers for more specific relative time definitions.

Time modifier	Description	Resulting time
@d-2h	Snap to the beginning of today (12 A.M.) and subtract 2 hours from that time.	10 P.M. last night.
-mon@mon+7d	One month ago, snapped to the first of the month at midnight, and add 7 days.	The 8th of last month at 12 A.M.

Examples of searches with relative time modifiers

Example 1: Web access errors from the beginning of the week to the current time of your search (now).

```
eventtype=webaccess error earliest=@w0
```

This search returns matching events starting from 12:00 A.M. of the Sunday of the current week to the current time. Of course, this means that if you run this search on Monday at noon, you will only see events for 36 hours of data.

Example 2: Web access errors from the current business week (Monday to Friday).

```
eventtype=webaccess error earliest=@w1 latest=+7d@w6
```

This search returns matching events starting from 12:00 A.M. of the Monday of the current week and ending at 11:59 P.M. of the Friday of the current week.

If you run this search on Monday at noon, you will only see events for 12 hours of data. Whereas, if you run this search on Friday, you will see events from the beginning of the week to the current time on Friday. The timeline however, will display for the full business week.

Example 3: Web access errors from the last full business week.

```
eventtype=webaccess error earliest=-7d@w1 latest=@w6
```

This search returns matching events starting from 12:00 A.M. of last Monday and ending at 11:59 P.M. of last Friday.

Specify time ranges for real-time searches

Time ranges for historical searches are set at the time the search runs. With real-time searches, the time range boundaries are constantly updating and by default, the results accumulate from the start of the search. You can also specify a range that represent a sliding window of time, for example, the last 30 seconds. When you specify a sliding window, Splunk software uses that amount of time to accumulate data. For example, if your sliding window is 5 minutes, you will not start to see data until after the first 5 minutes have passed. You can override this behavior so that Splunk software backfills the initial window with historical data before running in the normal real-time search mode. See Real-time backfill.

Real-time modifier syntax

To run a search in real time, you can select from predefined Real-time time range windows in the time range range picker list or you can specify a custom real-time window using **Custom time...** and selecting **Real-time**.

Time ranges for real-time search follow the same syntax as for historical searches, except that you precede the relative time specifier with "rt", so that it's rt<time_modifier>: `rt[+|-]<time_integer><time_unit>@<time_unit>`. See Specify time modifiers in your search.

These values are not designed to be used in a search string. If you are a Splunk Enterprise administrator, you can use these values when you edit the times.conf file (to add options to the time range picker), to specify the earliest/latest time ranges in the saved search dialog, or when you use the REST API to access the Splunk search engine.

When you use time range windows with real-time searches, some of the events that occur within the latest second may not be displayed. This is expected behavior and is due to the latency between the timestamps within the events and the time when the event arrives. Because the time range window is with respect to the timestamps within the events and not the time when the event arrives, events that arrive after the time window won't display.

Real-time searches over "all time"

There is a small difference between real-time searches that take place within a set time window (30 seconds, 1 minute, 2 hours) and real-time searches that are set to "all time."

- **In "windowed" real time searches,** the events in the search can disappear as they fall outside of the window, and events that are newer than the time the search job was created can appear in the window when they occur.
- **In "all-time" real-time searches,** the window spans all of your events, so events do not disappear once they appear in the window, but events that are newer than the time the search job was created can appear in the window as they occur.
- In comparison, **historical search** events never disappear from within the set range of time that you are searching and the latest event is always earlier than the job creation time (with the exception of searches that include events that have future-dated timestamps).

Real-time backfill

For windowed real-time searches, you can backfill the initial window with historical data. This is run as a single search, just in two phases: first, a search on historical data to backfill events; then, a normal real-time search. Real-time backfill ensures that real-time dashboards seeded with data on actual visualizations and statistical metrics over time periods are accurate from the start.

You can enable real-time backfill in the `limits.conf` file in the [realtime] stanza:

```
[realtime]

default_backfill = <bool>
* Specifies if windowed real-time searches should backfill events
* Defaults to true
```

See also

- About real-time searches and reports
- Expected performance and known limitations of real-time searches and reports

Use time to find nearby events

The Splunk Web timeline and time ranges for search are based on event **timestamps**.

While searching for errors or troubleshooting an issue, looking at events that happened around the same time can help correlate results and find the root cause. This topic discusses how you can search for surrounding events using an event's timestamp and using the timeline.

Use time accelerators

The `_time` field represents the timestamp of an event. When you run a search to retrieve events, the timestamp for each event is listed under the **Time** column.

You can click the timestamp of an event and open a dialog box containing controls, called a **_time accelerator**. Use the _time accelerator to run a new search that retrieves events chronologically close to that event.

You can search for all events that occurred before or after the event time. The accelerators are **Before this time**, **After this time**, and **At this time**. In addition, you can search for nearby events. For example, you can search for + 30 seconds, - 1 minutes, +/- 5 hours, and so on.

Use the timeline

The timeline is a histogram of the number of events returned by a Splunk search over a chosen time range. The time range is broken up into smaller time intervals (such as seconds, minutes, hours, or days), and the count of events for each interval is displayed as a column.

The location of each column on the timeline corresponds to an instance when the events that match your search occurred. If there are no columns at a time period, no events were found then. The taller the column, the more events occurred at that time.

Spikes in the number of events or no events along the timeline can indicate time periods that you want to investigate.

The timeline has **drilldown** functionality similar to the table and chart drilldown. When you click on a column in the timeline, your search results update to show only the events represented by the column. If you double-click on a column, you re-run the search over the time range represented by the column. Then, you can search for all surrounding events at this time range.

Subsearches

About subsearches

Using subsearches

A subsearch is a search within a primary, or outer, search. When a search contains a subsearch, the subsearch is run first.

Subsearches must be enclosed in square brackets in the primary search.

Consider the following search.

```
sourcetype=access_* status=200 action=purchase [search
sourcetype=access_* status=200 action=purchase | top limit=1 clientip |
table clientip] | stats count, dc(productId), values(productId) by
clientip
```

The subsearch portion of the search is enclosed in square brackets.

```
[search sourcetype=access_* status=200 action=purchase | top limit=1
clientip | table clientip]
```

The first command in a subsearch must be a **generating command** such as `search`, `eventcount`, or `tstats`. For a list of generating commands, see Command types in the *Search Reference*.

How subsearches work

A subsearch looks for a single piece of information that is then added as a criteria, or argument, to the primary search. You use a subsearch because the single piece of information that you are looking for is dynamic. The single piece of information might change every time you run the subsearch.

For example, you want to return all of the events from the host that was the most active in the last hour. The host that was the most active might be different from hour to hour. You need to identify the most active host before you can return the events from that host.

Break this search down into two parts.

- The most active host in the last hour. This is the subsearch.

- The events from that host. This is the primary search.

You could run two searches to obtain the list of events. The following search identifies the most active host in the last hour.

```
sourcetype=syslog earliest=-1h | top limit=1 host | fields host
```

This search returns only one host value. Assume that the result is the host named `crashy`. To return all of the events from the host `crashy`, you need to run a second search.

```
sourcetype=syslog host=crashy
```

The drawback to running two searches is that you cannot setup reports and dashboard panels to run automatically. You must run the first search to identify the piece of information that you need, and then run the second search with that piece of information.

You can combine these two searches into one search that includes a subsearch.

```
sourcetype=syslog [search sourcetype=syslog earliest=-1h | top limit=1
host | fields + host]
```

The subsearch is in square brackets and is run first. The subsearch in this example identifies the most active host in the last hour. The result of the subsearch is then provided as a criteria for the main search. The main search returns the events for the host.

Time ranges and subsearches

Time ranges that you specify directly in a search apply only to the main search. The time ranges specified in the main search do not apply to subsearches. Likewise, a time range specified in a subsearch applies only to that subsearch. The time range does not apply to the main search or any other subsearch.

Time ranges selected from the Time Range Picker apply to the main search and to subsearches.

When to use subsearches

Subsearches are mainly used for two purposes:

- Parameterize one search, using the output of another search. The example, described above, of searching for the most active host in the last hour is a an example of this use of a subsearch.
- Run a separate search and add the output to the first search using the `append` command.

A subsearch can be used only where the explicit action that you are trying to accomplish is with the search and not a transformation of the data. For example, you cannot use a subsearch with "`sourcetype=top | multikv`", because the `multikv` command does not expect a subsearch as an argument. Certain commands, such as `append` and `join` can accept a subsearch as an argument.

Multiple subsearches in a search string

You can use more than one subsearch in a search.

If a search has a set of nested subsearches, the inner most subsearch is run first, followed by the next inner subsearch, working out to the outermost subsearch and then the primary search.

For example, you have the following search.

```
index=* OR index=_* | [search index=* | stats count by component |
[search index=* | stats count by user | [search index=* | stats by
ipaddress]]]
```

The order in which the search is processed is:

1. `search index=* | stats by ipaddress`
2. `search index=* | stats count by user`
3. `search index=* | stats count by component`
4. (An implied search command) `index=* OR index=_*` (and the results of the nested subsearches)

Here is another example.

```
index=foo error [ search index=bar baz [search index=* | stats count by
user | search count>100] | stats count by host ]
```

Be certain to analyze your search syntax when you find yourself using subsearches frequently. It is often possible to rewrite the search and omit the subsearch.

If the subsearches are sequential instead of nested, the subsearch farthest to the left, or beginning of the search, is run first. Then working towards the right, or end of the search, the next subsearch is run. When all of the subsearches are run, then the primary search is run.

For example, you have the following search.

```
index=* OR index=_* | [search index=* | stats count by customerID] |
[search index=* | stats by productName]
```

The order in which the search is processes is:

1. `search index=* | stats by customerID`
2. `search index=* | stats count by productName`
3. (An implied search command) `index=* OR index=_*` (the results of the subsearches)

Subsearch examples

These examples show you the difference between searching your data with and without a subsearch.

These examples use the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to **get the tutorial data into Splunk**. Use the time range **All time** when you run the search.

Example 1: Without a subsearch, find what the most frequent shopper purchased

You want to find the single most frequent shopper on the Buttercup Games online store and what that shopper has purchased. Use the `top` command to return the most frequent shopper.

1. To find the shopper who accessed the online shop the most, use this search.

   ```
   sourcetype=access_* status=200 action=purchase | top limit=1
   clientip
   ```

   The `limit=1` argument specifies to return 1 value. The `clientip` argument specifies the field to return.

This search returns one `clientip` value, 87.194.216.51, which you will use to identify the VIP shopper.

2. You now need to run another search to determine how many different products the VIP shopper has purchased. Use the `stats` command to count the purchases by this VIP customer.

```
sourcetype=access_* status=200 action=purchase
clientip=87.194.216.51 | stats count, dc(productId),
values(productId) by clientip
```

This search uses the `count()` function to return the total count of the purchases for the VIP shopper. The `dc()` function is the distinct_count function. Use this function to count the number of different, or unique, products that the shopper bought. The `values` function is used to display the distinct product IDs as a multivalue field.

The drawback to this approach is that you have to run two searches each time you want to build this table. The top purchaser is not likely to be the same person at any given time range.

Example 2: Using a subsearch, find what the most frequent shopper purchased

Let's start with our first requirement, to identify the single most frequent shopper on the Buttercup Games online store.

1. Copy and paste the following search into the Search bar and run the search. Make sure the time range is **All time**.

   ```
   sourcetype=access_* status=200 action=purchase | top limit=1
   clientip | table clientip
   ```

   This search returns the clientip for the most frequent shopper, `clientip=87.194.216.51`. This search is almost identical to the search in Example 1 Step 1. The difference is the last piped command, `| table clientip`, which displays the clientip information in a table.

   To find what this shopper has purchased, you run a search on the same data. You provide the result of the most frequent shopper search as one of the criteria for the purchases search.

   The most frequent shopper search becomes the **subsearch** for the purchases search. The purchases search is referred to as the **outer** or primary search. Because you are searching the same data, the beginning of the outer search is identical to the beginning of the subsearch.

   A subsearch is enclosed in square brackets [ ] and processed first when the search is parsed.
2. Copy and paste the following search into the Search bar and run the search.

   ```
   sourcetype=access_* status=200 action=purchase [search
   sourcetype=access_* status=200 action=purchase | top limit=1
   clientip | table clientip] | stats count, dc(productId),
   values(productId) by clientip
   ```

   Because the `top` command returns the **count** and **percent** fields, the `table` command is used to keep only the `clientip` value.

These results should match the result of the two searches in Example 1, if you run it on the same time range. If you change the time range, you might see different results because the top purchasing customer will be different.

The performance of this subsearch depends on how many distinct IP addresses match `status=200 action=purchase`. If there are thousands of distinct IP addresses, the `top` command has to keep track of all of those addresses before the top 1 is returned, impacting performance. By default, subsearches return a maximum of 10,000 results and have a maximum runtime of 60 seconds. In large production environments, it is possible that the subsearch in this example will timeout before it completes. The best option is to rewrite the query to limit the number of events that the subsearch must process. Alternatively, you can increase the maximum results and maximum runtime parameters.

You can make the information more understandable by renaming the columns.

Column	Rename
count	Total Purchased
dc(productId)	Total Products
values(productId)	Product IDs
clientip	VIP Customer

You rename columns by using the AS operator on the fields in your search. If the rename that you want to use contains a space, you must enclose the rename in quotation marks.

3. To rename the fields, copy and paste the following search into the Search bar and run the search.

```
sourcetype=access_* status=200 action=purchase [search
sourcetype=access_* status=200 action=purchase | top limit=1
clientip | table clientip] | stats count AS "Total Purchased",
dc(productId) AS "Total Products", values(productId) AS "Product
IDs" by clientip | rename clientip AS "VIP Customer"
```

Subsearch performance considerations

A subsearch can be a performance drain if the search returns a large number of results.

Consider this search.

```
sourcetype=access_* status=200 action=purchase [search
sourcetype=access_* status=200 action=purchase | top limit=1 clientip |
table clientip] | stats count, dc(productId), values(productId) by
clientip
```

The performance of this subsearch depends on how many distinct IP addresses match `status=200 action=purchase`. If there are thousands of distinct IP addresses, the `top` command has to keep track of all of them before the top 1 is returned, impacting performance.

Additionally, by default subsearches return a maximum of 10,000 results and have a maximum runtime of 60 seconds. In large production environments it is quite possible that the subsearch in this example will timeout before it completes.

There are several alternatives you can use to control the results:

- Try to rewrite the query to limit the number of events the subsearch must process.
- You can change the number of results that the format command operates over inline with your search by appending the `format` command to the end of your subsearch.

  ```
  ...| format maxresults = <integer>
  ```

  For more information, see the `format` command in the *Search*

Reference.

If you are using Splunk Enterprise, you can also control the subsearch by editing settings in the `limits.conf` file. See How to edit a configuration file. Edit the settings for the runtime and maximum number of results returned.

[subsearch]
maxout = <integer>

- Maximum number of results to return from a subsearch.
- This value cannot be greater than or equal to 10500.
- Defaults to 10000.

maxtime = <integer>

- Maximum number of seconds to run a subsearch before finalizing
- Defaults to 60.

ttl = <integer>

- Time to cache a given subsearch's results, in seconds.
- Do not set this below 120 seconds.
- Defaults to 300.

After running a search you can click the **Job** menu and select *Inspect Job* to open the Search Job Inspector. Scroll down to the `remoteSearch` component, and you can see what the actual query that resulted from your subsearch. For more information, see View search job properties in this manual.

Output settings for subsearch commands

By default, subsearches return a maximum of 10,000 results. You will see variations in the actual number of output results because every command can change what the default `maxout` is when the command invokes a subsearch. Additionally, the default applies to subsearches that are intended to be expanded into a search expression, which is not the case for some commands such as join, append, and appendcols.

- For example, the `append` command can override the default maximum if the `maxresultrows` argument is specified, unless you specify `maxout` as an argument to the `append` command.

- The output limit of the `join` command is controlled by `subsearch_maxout` in the `limits.conf` file. This defaults to 50,000 events.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using subsearches.

Use subsearch to correlate events

A subsearch takes the results from one search and uses the results in another search. This enables sequential state-like data analysis. You can use subsearches to correlate data and evaluate events in the context of the whole event set, including data across different indexes or Splunk Enterprise servers in a distributed environment.

For example, you have two or more indexes for different application logs. The event data from these logs share at least one common field. You can use the values of this field to search for events in one index based on a value that is not in another index:

```
sourcetype=some_sourcetype NOT [search sourcetype=another_sourcetype |
fields field_val]
```

Note: This is equivalent to the SQL "NOT IN" functionality:

SELECT * from some_table
WHERE field_value
NOT IN (SELECT field_value FROM another_table)

Change the format of subsearch results

When you use a subsearch, the `format` command is implicitly applied to your subsearch results. The `format` command changes the subsearch results into a single linear search string. This is used when you want to pass the values in the returned fields into the primary search.

If your subsearch returned a table, such as:

```
| field1  | field2  |
```

```
            ------------------
event/row1 | val1_1  | val1_2  |
event/row2 | val2_1  | val2_2  |
```

The `format` command returns:

```
(field1=val1_1 AND field2=val1_2) OR (field1=val2_1 AND field2=val2_2)
```

For more information, see the format command.

Formatting exceptions

There are a couple of exceptions to the formatting that the `format` command performs.

- All internal fields, fields that begin with a leading underscore (_) character, are ignored and not formatted as a linear search string.
- If the name of a field is either `search` or `query`, the values of fields are rendered directly in the reformatted search string.

The search and query fields

You can rename a field to either `search` or `query` to change the format of the subsearch results. Renaming a field to `search` or `query` is a special use case. When you renaming your fields to anything else, the subsearch returns the new field names that you specify.

Using the search field name

Use the `search` field name and the `format` command when you need to append some static data or apply an evaluation on the data in the subsearch. You can then pass the data to the primary search. For example, you rename the second field in the search results to `search`, as shown in the following table:

```
           | field1  | search  |
            ------------------
event/row1 | val1_1  | val1_2  |
event/row2 | val2_1  | val2_2  |
```

Then using the `format` command returns:

```
(field1=val1_1 AND val1_2) OR (field1=val2_1 AND val2_2)
```

Instead of

```
(field1=val1_1 AND field2=val1_2) OR (field1=val2_1 AND field2=val2_2)
```

For multivalue fields, when you use the `search` field name, the first value of the field is used as the actual search term.

Using the query field name

Use the `query` field name when you want the values in the fields returned from the subsearch, but not the field names.

The `query` field name is similarly to using the `format` command. Instead of passing the field and value pairs to the main search, such as:

```
(field1=val1_1 AND field2=val1_2) OR (field1=val2_1 AND field2=val2_2)
```

Using the `query` field name passes only the values:

```
(val1_1 AND val1_2) OR (val2_1 AND val2_2)
```

Examples

The following search looks for a value in the `clID` field that is associated with a name token or field value. The clID value is then used to search for several sources.

```
index=myindex [search index=myindex host=myhost MyName | top limit=1
clID | fields clID ]
```

The subsearch returns the field and value in the format: `( (clID="0050834ja") )`

To return only the value, `0050834ja`, rename the `clID` field to `search` in the subsearch. For example:

```
index=myindex [search index=myindex host=myhost MyName | top limit=1
clID | fields clID | rename clID as search ]
```

When the field is named `search` or `query`, the field name is dropped and the implicit `| format` command at the end of the subsearch returns only the value.

If you return multiple values, such as specifying `...| top limit=3`, the subsearch returns each of the values with the boolean OR operator between the values. For example, if the previous search example used `...| top limit=3`, the values returned from the subsearch are `( ( value1 ) OR ( value2 ) OR ( value3 ) )`.

Create Statistical Tables and Chart Visualizations

About transforming commands and searches

To create charts visualizations, your search must transform event data into statistical data tables. These statistical tables are required for charts and other kinds of data visualizations. This section discusses how to use **transforming commands** to transform event data.

This section describes the major categories of transforming commands and provides examples of how they can be used in a search.

Transforming commands

The primary transforming commands are:

- `chart`: creates charts that can display any **series** of data that you want to plot. You can decide what field is tracked on the x-axis of the chart.
- `timechart`: used to create "trend over time" reports, which means that `_time` is always the x-axis.
- `top`: generates charts that display the most common values of a field.
- `rare`: creates charts that display the least common values of a field.
- `stats`: generates a report that display summary statistics.

See Transforming commands in the *Search Reference* to learn more.

Note: As you will see in the following examples, you always place your transforming commands after your search commands, linking them with a **pipe operator** (|).

The `chart`, `timechart`, and `stats` commands are all designed to work with statistical functions. The list of available statistical functions includes:

- count, distinct count
- mean, median, mode
- min, max, range, percentiles
- standard deviation, variance
- sum

- first occurrence, last occurrence

For more information about statistical functions, see Statistical and charting functions in the *Search Reference*. Some statistical functions only work with the `timechart` command.

Note: All searches with transforming commands generate specific data structures. The different chart types require these data structures to be set up in particular ways. For example, not all searches that enable you to generate bar, column, line, and area charts can be used to generate pie charts. See Data structure requirements for visualizations in the *Dashboard and Visualizations* manual to learn more.

Table, chart, and report examples

The following examples use transforming commands to create tables, charts, and reports:

- Create time based charts
- Create charts that are not (necessarily) time-based
- Create reports that display summary statistics
- Build a chart of multiple data series

Real-time reporting

You can use real-time search to calculate metrics in real time on large incoming data flows without the use of summary indexing. However, because you are reporting on a live and continuous stream of data, the timeline will update as the events stream in and you can only view the table or chart in preview mode. Also, some search commands will be more applicable (for example, streamstats and rtorder) for use in real-time. See About real-time searches and reports.

See also

Types of commands
Types of searches

Create time-based charts

This topic discusses using the timechart command to create time-based reports.

The timechart command

The timechart command generates a table of summary statistics. This table can then be formatted as a chart visualization, where your data is plotted against an x-axis that is always a time field. Use the timechart command to display statistical trends over time You can split the data with another field as a separate series in the chart. Timechart visualizations are usually line, area, or column charts.

When you use the `timechart` command, the x-axis represents time. The y-axis can be any other field value, count of values, or statistical calculation of a field value.

For more information, see the Data structure requirements for visualizations in the *Dashboards and Visualizations* manual.

Examples

Example 1: This report uses internal Splunk log data to visualize the average indexing thruput (indexing kbps) of Splunk processes over time. The information is separated, or split, by processor:

```
index=_internal "group=thruput" | timechart avg(instantaneous_eps) by
processor
```

See also

Build a chart of multiple data series

Create charts that are not (necessarily) time-based

This topic discusses using the **transforming command**, chart, to create visualizations that are not time-based.

The chart command

The chart command returns your results in a data structure that supports visualization of your data series as a chart such as a column, line, area, and pie chart.

Unlike the `timechart` command, which uses the `_time` **default field** as the x-axis, charts created with the `chart` command use an arbitrary field as the x-axis. With

the chart command, you use the `over` keyword to determine what field takes the x-axis.

Examples

Example 1: Use web access data to show you the average count of unique visitors over each weekday.

```
sourcetype=access_* | chart avg(clientip) over date_wday
```

One of the options you have is to split the data by another field, meaning that each distinct value of the "split by" field is a separate series in the chart. If your search includes a "split by" clause, place the `over` clause before the "split by" clause.

The following report generates a chart showing the sum of kilobytes processed by each `clientip` within a given timeframe, split by `host`. The finished chart shows the `bytes` value taking the y-axis while `clientip` takes the x-axis. The delay value is broken out by host. After you run this search, format the report as a stacked bar chart.

```
sourcetype=access_* | chart sum(bytes) over clientip by host
```

Example 2: Create a stacked bar chart that splits out the http and https requests hitting your servers.

To do this, first create `ssl_type`, a search-time **field extraction** that contains the inbound port number or the incoming URL request, assuming that it is logged. The finished search would look like this:

```
sourcetype=access_* | chart count over ssl_type
```

After you run the search, format the results as a stacked bar chart.

Visualize field value highs and lows

This topic discusses how to use the **transforming commands**, top and rare, to create charts that display the most and least common values.

The top and rare commands

The top command returns the most frequent values of a specified field in your returned events. The rare command, returns the least common value of a specified field in your returned events. Both commands share the same syntax. If you don't specify a limit, the default number of values displayed in a `top` or `rare` is ten.

Examples

Example 1: Generate a report that sorts through firewall information to list the top 100 destination ports used by your system:

```
sourcetype=firewall | top limit=100 dst_port
```

Example 2: Generate a report that shows you the source ports with the lowest number of denials.

```
sourcetype=firewall action=Deny | rare src_port
```

A more complex example of the top command

Say you're indexing an alert log from a monitoring system, and you have two fields:

- `msg` is the message, such as `CPU at 100%`.
- `mc_host` is the host that generates the message, such as `log01`.

How do you get a report that displays the top `msg` and the values of `mc_host` that sent them, so you get a table like this:

Messages by mc_host
CPU at 100%
log01
log02
log03
Log File Alert
host02

host56
host11

To do this, set up a search that finds the top `message` per `mc_host` (using `limit=1` to only return one) and then `sort` by the message `count` in descending order:

```
sourcetype=alert_log | top 1 msg by mc_host | sort count
```

Create reports that display summary statistics

This topic discusses using the stats and eventstats **transforming commands** to create reports that display summary statistics related to a field.

The stats and eventstats commands

The `eventstats` command works in exactly the same manner as the `stats` command, except that the aggregation results of the command are added inline to each event, and only the aggregations that are pertinent to each event.

Using split-by clauses

To fully utilize the `stats` command, you need to include a "split by" clause. For example, the following report won't provide much information:

```
sourcetype=access_combined | stats avg(kbps)
```

It gives you the average of `kbps` for all events with a sourcetype of `access_combined`--a single value. The resulting column chart contains only one column.

But if you break out the report with a split by field, Splunk software generates a report that breaks down the statistics by that field. The following report generates a column chart that sorts through the `access_combined` logs to get the average thruput (kbps), broken out by host:

```
sourcetype=access_combined | stats avg(kbps) by host
```

Examples

Example 1: Create a report that shows you the CPU utilization of Splunk processes, sorted in descending order:

```
index=_internal "group=pipeline" | stats sum(cpu_seconds) by processor
| sort sum(cpu_seconds) desc
```

Example 2: Create a report to display the average kbps for all events with a sourcetype of `access_combined`, broken out by host.

You specify the field name for the `eventstats` results by adding the `as` argument. So the first example above could be restated with "avgkbps" being the name of the new field that contains the results of the `eventstats avg(kbps)` operation:

```
sourcetype=access_combined | eventstats avg(kbps) as avgkbps by host
```

When you run this set of commands, Splunk software adds a new `avgkbps` field to each `sourcetype=access_combined` event that includes the `kbps` field. The value of `avgkbps` is the average kbps for that event.

Look for associations, statistical correlations, and differences in search results

This topic discusses **transforming commands** that find associations, similarities, and differences among field values in your search results.

The associate command

The associate command identifies events that are associated with each other through field/field value pairs. For example, if one event has a `referer_domain` of "http://www.google.com/" and another event has a `referer_domain` with the same URL value, then they are associated.

"Tune" the results gained by the `associate` command with the *supcnt*, *supfreq*, and *improv* arguments. For more information about these arguments see the associate command reference topic.

Example: Search the web access sourcetypes and identify events that share at least three field/field-value pair associations.

```
sourcetype=access* | associate supcnt=3
```

The correlate command

The correlate command calculates the statistical correlation between fields. It uses the `cocur` operation to calculate the percentage of times that two fields exist in the same set of results.

Example:' Search across all events where `eventtype=goodaccess`, and calculates the co-occurrence correlation between all of those fields.

```
eventtype=goodaccess | correlate type=cocur
```

The diff command

Use the diff command to compare the differences between two search results. By default it compares the raw text of the search results you select, unless you use the *attribute* argument to focus on specific field attributes.

Example: Compare the IP addresses for the 44th and 45th events returned in the search.

```
eventtype=goodaccess | diff pos1=44 pos2=45 attribute=ip
```

Build a chart of multiple data series

Splunk **transforming commands** do not support a direct way to define multiple data **series** in your charts (or timecharts). However, you CAN achieve this using a combination of the `stats` and `xyseries` commands.

The `chart` and `timechart` commands both return tabulated data for graphing, where the x-axis is either some arbitrary field or `_time`, respectively. When these commands are used with a split-by field, the output is a table where each column represents a distinct value of the split-by field.

In contrast, the `stats` command produces a table where each row represents a single unique combination of the values of the group-by fields. You can then use the `xyseries` command to redefine your data series for graphing.

For most cases, you can simulate the results of "... | chart n by x,y" with "... | stats n by x,y | xyseries x y n". (For the `timechart` equivalent of results, x = `_time`.)

Scenario

Let's say you want to report on data from a cluster of application servers. The events gathered from each server contain information such as counts of active sessions, requests handled since last update, etc. and are placed in the `applications_servers` index. You want to display each server instance and the number of sessions per instance on the same timechart so that you can compare the distributions of sessions and load.

Ideally, you want to be able to run a timechart report, such as:

```
index=application_servers | timechart sum(handledRequests)
avg(sessions) by source
```

However, timechart does not support multiple data series; so instead, you need run a search similar to the following:

```
index=application_servers | bin _time | stats sum(handledRequests) as
hRs, avg(sessions) as ssns by _time,source | eval s1="handledReqs
sessions" | makemv s1 | mvexpand s1 | eval
yval=case(s1=="handledReqs",hRs,s1=="sessions",ssns) | eval
series=source+":"+s1 | xyseries _time,series,yval
```

Walkthrough

```
... | bin _time
```

The first thing that you need to do, before the `stats` command, is to separate the events by time.

```
... | stats sum(handledRequests) as hRs, avg(sessions) as ssns by
_time,source
```

The `stats` command is used to calculate statistics for each source value: The sum of `handledRequests` values are renamed as `hRs`, and the average number of `sessions` are renamed as `ssns`.

```
... | eval s1="handledRequests sessions" | makemv s1 | mvexpand s1
```

This uses the `eval` command to add a single-valued field "s1" to each result from the stats command. Then, the makemv command converts s1 into a multivalued field, where the first value is "handledRequests" and the second value is "sessions". The mvexpand then creates separate series for each value of s1.

```
... | eval yval=case(s1=="handledRequests",hRs,s1=="sessions",ssns)
```

This uses the eval command to define a new field, yval, and assign values to it based on the case that it matches. So, if the value of s1 is "handledRequests", yval is assigned the "hRs" value. And, if the value of s1 is "sessions", yval is assigned the "ssns" value.

```
... | eval series=source+":"+s1
```

This uses the eval command to define a new field, series, which concatenates the value of the host and s1 fields.

```
... | xyseries _time,series,yval
```

Finally, the xyseries command is used to define a chart with _time on the x-axis, yval on the y-axis, and data defined by series.

Compare hourly sums across multiple days

The `timechart command` creates charts that show trends over time. It has strict boundaries limiting what it can do. There are times when you should use the `chart command` command, which can provide more flexibility.

This example demonstrates how to use `chart` to compare values collected over several days. You cannot do this with `timechart`

Scenario

These two searches are almost identical. They both show the hourly sum of the `P` field over a 24-hour period. The only difference is that one search covers a period ten days in the past, while the other covers a period nine days into the past:

Search 1:

```
earliest=-10d latest=-9d | timechart span="1h" sum(P)
```

Search 2:

```
earliest=-9d latest=-8d | timechart span="1h" sum(P)
```

Create a column chart that combines the results of these two searches, so you can see the sum of `P` for 3pm, ten days ago side-by-side with the sum of `P` for 3pm, nine days ago.

Solution

Using the chart command, set up a search that covers both days. Then, create a "sum of P" column for each distinct `date_hour` and `date_wday` combination found in the search results.

The finished search looks like this:

```
earliest=-10d latest=-8d | chart sum(P) by date_hour date_wday
```

This produces a single chart with 24 slots, one for each hour of the day. Each slot contains two columns that enable you to compare hourly sums between the two days covered by the time range of the report.

For a primer on reporting searches and how they're constructed, see "Use reporting commands" in the *Search Manual*.

For more information about `chart>` and `timechart` functions, see "Statistical and charting functions" in the *Search Reference*.

Drill down on tables and charts

After running a search, you can run different kinds of secondary **drilldown** searches. The drilldown search options depend on the type of element you click on.

In the **Statistics** tab you can run a drilldown search when you click on a field value or calculated search result.

For information on drilling down on field-value pairs, see Drill down on event details.

You can also click on elements of charts and visualizations to run drilldown searches. For more information see Use drilldown for dashboard interactivity in *Dashboards and Visualizations*.

Open a non-transforming search in Pivot to create tables and charts

Searches that do not contain **transforming commands** return event lists that you can view in the **Events** tab, and you can use the **Patterns** tab to see the dominant patterns amongst those events. However, these non-transforming searches cannot return results in the form of statistical tables. Without statistical tables, Splunk software cannot create charts or other visualizations. This means that when you run a non-transforming search you do not get results in the **Statistics** or **Visualization** tabs.

If you run a non-transforming search and want to make tables or charts based on it, go to the **Statistics** or **Visualization** tab and open the search in Pivot.

In the **Pivot Editor**, you can build tables and charts without editing the original search string. As you work with the Pivot Builder to refine your visualizations, the underlying search is rerun as required so you can see the effect of your changes.

When you save a visualization that you create in the Pivot Editor as a report or dashboard panel, a corresponding **data model** is created. This data model is the foundation of the saved report or dashboard panel. It defines the underlying search and the fields involved in the report or dashboard panel. Without it you cannot rerun the report or view the panel that you saved.

Open a search in Pivot

1. In the Search view, run a non-transforming search. For example:
   ```
   sourcetype=access_* status=200 action=purchase
   ```
2. Go to the **Statistics** or **Visualization** tab and click **Pivot.**
3. Select the set of fields that you want to use to build your pivot table or chart in the Pivot Builder.

   Each option displays the number of fields it represents in parenthesis:

   All Fields provides all of the fields that were discovered by the search.

   Selected Fields restricts you to the fields identified as **Selected Fields** for the search on the **Fields** tab. If you open a search in Pivot without making changes or selecting fields, the **Selected Fields** option provides the default selected fields: `host`, `source`, and `sourcetype`. To build your pivot table or chart using a different set of fields, go to the **Fields** tab and select the fields in the Selected Fields list

before you move to the **Statistics** or **Visualization** tab and open the search in Pivot.

Fields with at least lets you set a coverage threshold for your fieldset. For example, to work with fields that apply to the majority of your events, set the threshold to something high, like 70%. The fieldset you get in Pivot only includes fields that exist in 70% (or more) of the events returned by the search.

4. Click **Ok** to go to the **Pivot Editor**.
5. Build your pivot table or chart.

   The **Attributes** list in the pivot element types (filters, split rows, split columns, and column values) contains the fieldset that you selected in step 3.

   Note: If you navigate away from the Pivot Editor without saving your table or chart, your work is lost.

   To save your work, see the next subtopic, "Save the finished pivot table or chart."

While in the Pivot Editor, you can click **Open in Search** to open the pivot search in the Search interface and edit that search. This action takes you out of the Pivot Editor and prevents you from saving any pivot table or chart you created (see the next subtopic). For more information about using the Pivot Editor to design tables, see "Design pivot tables with the Pivot Editor." For more information about using the Pivot Editor to design charts and other visualizations, see "Design pivot charts and visualizations with the Pivot Editor."

Save the finished pivot table or chart

You can save a table or chart in the Pivot Editor as a report or dashboard panel. However, Splunk software must also create a data model to support the saved report or panel. This data model is required to access the report or panel after you have saved it.

1. In the Pivot Editor, click **Save As** and select either *Report* or *Dashboard Panel*.

   Depending on which one you choose, either the Save As Report or Save As Dashboard Panel dialog box appears.
2. In the **Save As** dialog box provide information for the report or dashboard panel that you are saving.

   For more information about these fields, see the documentation on saving reports or saving dashboard panels.

3. In the **Save as** dialog box type the **Model Name** and **Model ID** for the data model that will support the report or dashboard panel.
   You can manage the model that Splunk Enterprise creates through this process if your role has admin-level capabilities.
4. Click **Save** to save the report or dashboard panel and create the data model.
   Click a button to view the new report or dashboard panel, or go to the new data model by clicking the name of the model.
   Click the data model name to go to the Data Model Builder, where you can change the fields associated with the model and add data model dataset to the model.

About permissions for datasets created though this method

Newly created data models are private and can only be seen and used by the person who created them. Only users with admin or power roles (or a role with equivalent permissions) can share data models. If the data model is not shared, reports or dashboard panels created with that data model cannot be shared either. In addition, data models cannot be accelerated until they are shared.

If you have created a report or dashboard panel that is for your use only, you do not have to do anything. If you want other users to be able to access the report or dashboard panel, share the related data model, if you have appropriate permissions, or have a user with admin-level permissions share it for you.

For more information about data model permissions see "Manage data models" in the *Knowledge Manager Manual*.

Search and Report in Real Time

About real-time searches and reports

With **real-time searches** and reports, you can search events before they are **indexed** and preview reports as the events stream in.

- You can design alerts based on real-time searches that run continuously in the background. Such **real-time alerts** can provide timelier notifications than alerts that are based on **scheduled reports**. For more information, see the *Alerting Manual*.
- You can also display real-time search results and reports in **dashboards**. For more information, see the dashboard overview in *Dashboards and Visualizations*.

The number of concurrent real-time searches can greatly affect indexing performance. To lessen the impact on the indexer, you can enable indexed real-time searches, which is described later in this topic. See Expected performance and known limitations of real-time searches and reports.

By default, only users with the Admin role can run and save real-time searches. For more information on managing roles and assigning them to users, see Add and edit roles in *Securing Splunk Enterprise*.

Real-time search mechanics

Real-time searches scan events as the events arrive for indexing. When you kick off a real-time search, Splunk software scans the incoming events. The scan looks for events that contain index-time fields that indicate the event *could* be a match for your search.

As the real-time search runs, the software periodically evaluates the scanned events against your search criteria to find actual matches within the sliding **time range window** that you have defined for the search. The number of matching events can fluctuate up or down over time as the search discovers matching events at a faster or slower rate. If you are running the search in Splunk Web, the search timeline also displays the matching events that the search has returned within the chosen time range.

Here is an example of a real-time search with a one minute time range window. At the point that the following screen capture was taken, the search had scanned

a total of 436 events since it was launched. The matching event count of 333 represents the number of events matching the search criteria that were identified in the past minute. This number fluctuated between 312 and 357 for the following minute. If the number spiked or dropped dramatically, that could indicate that something interesting was happening that requires a closer look.

As you can see, the newest events are on the right-hand side of the timeline. As time passes, the events move left until the events move off the left-hand side, disappearing from the time range window entirely.

A real-time search should continue running until you or another user stops the search or deletes the search job. The real-time search should not "time out" for any other reason. If your events are stopping it could be a performance-related issue (see "Expected performance and known limitations").

Real-time searches can take advantage of all search functionality, including advanced functionality like lookups, transactions, and so on. There are also search commands that are to be used specifically in conjunction with real-time searches, such as `streamstats` and `rtorder`.

Indexed real-time search

Enabling your real-times searches to run after the events are indexed can greatly improve indexing performance. This is especially true if there are a large number of concurrent real-time searches. To lessen the impact on the indexer, you can enable indexed real-time search. This runs searches like **historical searches**, but also continually updates the search with new events as the events appear on disk.

Use indexed real-time search when up-to-the-second accuracy is not needed.

Prerequisites

- Only users with file system access, such as system administrators, can enable indexed real-time search.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

Steps

1. Open the local `limits.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Under the `[realtime]` stanza, set `indexed_realtime_use_by_default` to `true`.

If you are using Splunk Cloud and want to change the default to indexed real-time search, open a Support ticket.

About the sync delay lag time

The results returned by an *indexed* real-time search will always lag behind a real-time search. Built into indexed real-time searches is a sync (synchronizing) delay. The sync delay is a precaution so that none of the data is missed.

Indexed data does not necessarily appear on disk in the order that the data is indexed because:

- Multiple threads are used for indexing simultaneously
- The sync delay ordering that is on your operating system

An indexed real-time must remember the latest indexed event that is returned for the current iteration of the time range window. That event is used as the start point for the next iteration of the time range window. If a sync delay is not imposed, some of the events before the latest event might not be searchable yet. These events are not returned during that iteration of the time range window and will never be returned. The likelihood of an unreturned event increases as the indexing and system load increases.

You can control the number of seconds of sync delay lag time with the `indexed_realtime_disk_sync_delay = <int>` setting. By default, this delay is set to 60 seconds.

The default of 60 seconds is fairly conservative. For most systems a 30 second delay will probably work successfully. If, for your system and usage, it is acceptable for indexed real-time searches to miss some events, you can set a very low or 0 sync delay. However, you will not be able to tell if you are missing events, except for searches that should match all events.

Other indexed real time settings

There are other settings that you can use to configure indexed real-time search behavior, including:

- indexed_realtime_default_span
- indexed_realtime_maximum_span
- indexed_realtime_cluster_update_interval

These settings are described in the limits.conf.spec file.

See also

- Real-time searches and reports in Splunk Web
- Real-time searches and reports in the CLI
- Expected performance and known limitations of real-time searches and reports
- How to restrict usage of real-time searches

Blogs

- Splunk: Limiting Real-Time Searches and Maximizing Performance Gainz by Hurricane Labs

Real-time searches and reports in Splunk Web

Real-time searches in Splunk Web

You run a **real-time search** in exactly the same way you run **historical searches**. However, because you are searching a live and continuous stream of data, the timeline updates as the events stream in and you can only view the

report in preview mode. Also, some search commands are more applicable to real-time searches than historical searches. For example, streamstats and rtorder were designed for use in real-time searches.

To kick off a real-time search in Splunk Web, use the time range menu to select a preset real-time **time range window**, such as **30 seconds** or **1 minute.** You can also specify a sliding time range window to apply to your real-time search.

If you have Apache web access data, run the following search to see web traffic events as they stream in.

```
sourcetype=access_*
```

The raw events that are streamed from the input pipeline are not time-ordered. You can use the `rtorder` command to buffer the events from a real-time search and emit them in ascending time order.

The following example keeps a buffer of the last 5 minutes of web traffic events, emitting events in ascending time order once they are more than 5 minutes old. Newly received events that are older than 5 minutes are discarded if an event after that time has already been emitted.

```
sourcetype=access_* | rtorder discard=t buffer_span=5m
```

Real-time search relies on a stream of events. Thus, you cannot run a real-time search with any other leading search command, such as `| metadata` which does not produce events or `| inputcsv` which just reads in a file. Also, if you try to send the search results to `| outputcsv`, the CSV file will not be written until the real-time search is Finalized.

Real-time reports in Splunk Web

Run a report to preview the IP addresses that access the most web pages. In this case, the `top` command returns a table with three columns: clientip, count, and percent. As the data streams in, the table updates with new values.

```
sourcetype=access_* | top clientip
```

For each web traffic event, add a `count` field that represents the number of events seen so far (but do not include the current event in the count).

```
sourcetype=access_* | streamstats count current=false
```

You can also drilldown into real-time reports. However, real-time drilldown does not spawn another real-time search. Instead, it spawns a historic search, as you will drilldown into the events that have already been retrieved and indexed. For more information, see Use drilldown for dashboard interactivity in *Dashboards and Visualizations*.

See also

- About real-time searches and reports
- Real-time searches and reports in the CLI
- Expected performance and known limitations of real-time searches and reports
- How to restrict usage of real-time searches

Real-time searches and reports in the CLI

To run a real-time search in the CLI, replace the command "search" with "rtsearch":

./splunk rtsearch 'eventtype=pageview'

Use the `highlight` command to emphasize terms in your search results. The following example highlights "GET" in your page view events:

./splunk rtsearch 'eventtype=pageview | highlight GET'

By default, search results have line wrapping enabled. Use the `-wrap` option to turn off line wrapping:

./splunk rtsearch 'eventtype=pageview' -wrap 0

Real-time reports in the CLI will also display in preview mode and update as the data streams in.

./splunk rtsearch 'error | top clientip'

Use the `-preview` option to suppress the results preview:

./splunk rtsearch 'error | top clientip' -preview false

If you turn off preview, you can still manage (Save, Pause, Finalize, or Delete) the search from the Jobs page in Splunk Web. After you finalize the search, the report table will display. For more information, see "Supervise jobs with the Jobs page" in this manual.

To run a windowed real-time search, use the `earliest_time` and `latest_time` parameters.

rtsearch 'index=_internal' -earliest_time 'rt-30s' -latest_time 'rt+30s'

Note: Real-time searches can only be set at the API level, so the search does not run if you try to specify the time range modifiers within the search string. The `earliest_time` and `latest_time` parameters should set the same-name arguments in the REST API.

You can view all CLI commands by accessing the CLI help reference. For more information, see "Get help with the CLI" in this manual.

See also

- About real-time searches and reports
- Real-time searches and reports in Splunk Web
- Expected performance and known limitations of real-time searches and reports
- How to restrict usage of real-time searches

Expected performance and known limitations of real-time searches and reports

Real-time search matches events that have arrived at the port but have not been persisted to disk. The rate of arrival of events and number of matches can determine how much memory is consumed and the impact on the indexing rate.

Indexing throughput

Splunk software performance is expected to be acceptable as long as the indexers are not currently heavily loaded and do not have more than a few concurrent real-time searches.

Real-time searches will have a significant impact on performance in high volume environments and network load when you have many concurrent real-time

searches.

When planning your real-time searches, you should consider how it will affect the performance of both:

- The **search peer** that must stream the live events.
- The **search head** that must process the aggregated stream of live events.

The more work that is done on the search peer, the less that is required on the search head, and vice versa. The search peer is important to the overall system function, so you do not want to burden it with too much filtering of live events. However, if the search peer does not filter at all, the processing power and bandwidth required to send all the live events to the search head may prove costly, especially when you have multiple real-time searches running concurrently.

In cases where the search head cannot keep up with the search peer, the queue on the index processor will cease to flag events for the search. However, the events will have a sequence number that you can use to tell when and how many events were omitted from search consideration.

Concurrent real-time and historical searches

You can run real-time and historical searches concurrently, within the limits of your hardware. There are no restrictions on separate searches for the same or different users.

Concurrent real-time searches

Running multiple real-time searches will negatively impact indexing capacity. The real-time search feature is optimized for real-time alerting on sparse, or rare-term, searches and sacrifices indexing capacity for improved latency and reliability.

Indexed real-time searches

The number of concurrent real-time searches can greatly affect indexing performance. To lessen the impact on the indexer, you can enable indexed real-time search. This will basically run the search like a historical search, but will also continually update it with new events as they appear on disk.

Read more about how to enable indexed real-time search in About real-time searches and reports.

Real-time search windows

Windowed real-time searches are more expensive than non-windowed. The operations required to manage and preview the window contents can result in a windowed real time search not keeping up with a high rate of indexing. If your windowed search does not display the expected number of events, try a non-windowed search. If you are interested only in event counts, try using "timechart count" in your search.

See Specify time ranges for real-time searches.

See also

- About real-time searches and reports
- Real-time searches and reports in Splunk Web
- Real-time searches and reports in the CLI
- How to restrict usage of real-time searches

How to restrict usage of real-time search

Because overuse of real-time search can result in performance costs, you may find it necessary to restrict its usage.

Options for restricting real-time search are as follows:

- Disable real-time search at the indexer level by editing `indexes.conf` for specific indexes.
- Disable real-time search for particular roles and users.
- Edit `limits.conf` to reduce the number of real-time searches that can be run concurrently at any given time.
- Edit `limits.conf` to restrict indexer support for real-time searches.

If you are using Splunk Cloud and want to restrict real-time search, file a Support ticket.

Disable real-time search in indexes.conf

Searching in real time may be very expensive on the indexer. If you want to disable it on an indexer, you can edit a `[default]` setting in that indexer's `indexes.conf`. Note that this setting cannot be overridden on an index-by-index basis, it applies to all indexes located on the indexer.

```
[default]
enableRealtimeSearch = <bool>
```

Note: A *search head* that connects to multiple indexers will still be able to get real-time search results from the indexers that do have it enabled.

Disable real-time search for a user or role

Real-time search is a **capability** that you can map to specific users or roles in Splunk Web from **Manager > Access Controls**. By default, the **rtsearch** capability is assigned to the Admin and Power roles and not the User role. A role without the rtsearch capability will not be able to run a real-time search on that search head, regardless what indexers that search head is connected to.

Set search limits on real-time searches

You can use the `[search]` stanza in `limits.conf` to change the maximum number of real-time searches that can run concurrently on your system.

```
[search]
max_rt_search_multiplier = <decimal number>
realtime_buffer = <int>
```

`max_rt_search_multiplier`

- A number by which the maximum number of historical searches is multiplied to determine the maximum number of concurrent real-time searches. Defaults to 1.
- **Note:** The maximum number of real-time searches is computed as:
  `max_rt_searches = max_rt_search_multiplier x max_hist_searches`

`realtime_buffer`

- The maximum number of accessible events to keep for real-time searches from the UI. Must be >= 1. Defaults to 10000.
- The real-time buffer acts as a circular buffer once this limit is reached.

Set indexer limits for real-time search

You can use the `[realtime]` stanza in `limits.conf` to change the default settings for indexer support of real-time searches. **These options can be overridden for individual searches via REST API arguments.**

```
[realtime]
queue_size = <int>
blocking = [0|1]
max_blocking_secs = <int>
indexfilter = [0|1]

queue_size = <int>
```

- The size of queue for each real-time search. Must be > 0.
- Defaults to 10000.

```
blocking =[0|1]
```

- Specifies whether the indexer should block if a queue is full.
- Defaults to false (0).

```
max_blocking_secs = <int>
```

- The maximum time to block if the queue is full. This option is meaningless, if `blocking = false`.
- Means "no limit" if set to 0.
- Defaults to 60.

```
indexfilter = [0|1]
```

- Specifies whether the indexer should pre-filter events for efficiency.
- Defaults to true (1).

See also

- About real-time searches and reports
- Real-time searches and reports in Splunk Web
- Real-time searches and reports in the CLI
- Expected performance and known limitations of real-time searches and reports

Evaluate and Manipulate Fields

About evaluating and manipulating fields

This section discusses the search commands that enable you to evaluate new fields, manipulate existing fields, enrich events by adding new fields, and parse fields with multiple values.

- At the core of evaluating new fields is the `eval` command and its functions. Unlike the `stats` command, which enables you to calculate statistics based on fields in your events, the `eval` command enables you to create new fields using existing fields and an arbitrary expression. The `eval` command has many functions. See Use the eval command and functions.
- You can easily enrich your data with more information at search time. See Use lookup to add fields from external lookup tables.
- You can use the Splunk SPL (search processing language) to extract fields in different ways using a variety of search commands.
- Your events might contain fields with more than one value. There are search commands and functions that work with multivalue fields. See Manipulate and evaluate fields with multiple values.

Use the eval command and functions

The eval command enables you to devise arbitrary expressions that use automatically extracted fields to create a new field that takes the value that is the result of the expression's evaluation. The eval command is immensely versatile and useful. But while some eval expressions are relatively simple, they often can be quite complex.

This topic discusses how to use the eval command and the evaluation functions.

Types of eval expressions

An eval expression is a combination of literals, fields, operators, and functions that represent the value of your destination field. The expression can involve a mathematical operation, a string concatenation, a comparison expression, a boolean expression, or a call to one of the eval functions. Eval expressions require that the field's values are valid for the type of operation.

For example, with the exception of addition, arithmetic operations may not produce valid results if the values are not numerical. For addition, eval can concatenate the two operands if they are both strings. When concatenating values with '.', eval treats both values as strings, regardless of their actual type.

Example 1: Use an eval expression with a stats function.

Search all indexes and count the number of events where the `status` field value is `404`. Rename the results to a field called `count_status` and organize the results by source type.

```
index=* | stats count(eval(status="404")) as count_status by sourcetype
```

Example 2: Define a field that is the sum of the areas of two circles

Use the `eval` command to define a field that is the sum of the areas of two circles, A and B.

```
... | eval sum_of_areas = pi() * pow(radius_a, 2) + pi() * pow(radius_b,
2)
```

The area of circle is r^2, where r is the radius. For circles A and B, the radii are radius_a and radius_b, respectively. This eval expression uses the `pi` and `pow` functions to calculate the area of each circle and then adds them together, and saves the result in a field named, `sum_of_areas`.

Example 3: Define a location field using the city and state fields

Use the `eval` command to define a location field using the city and state fields. For example, if the city=Philadelphia and state=PA, location="Philadelphia, PA".

```
... | eval location=city.", ".state
```

This eval expression is a simple string concatenation.

Example 4: Use eval functions to classify where an email came from

This example uses sample email data. You should be able to run this search on any email data by replacing the `sourcetype=cisco:esa` with the `sourcetype` value and the `mailfrom` field with email address field name in your data. For example, the email might be `To`, `From`, or `Cc`).

This example classifies where an email came from based on the email address domain. The .com, .net, and .org addresses are considered **local**, while anything else is considered **abroad**. There are many domain names. Of course, domains that are not .com, .net, or .org are not necessarily from **abroad**. This is just an example.

The `eval` command in this search contains multiple expressions, separated by commas.

```
sourcetype="cisco:esa" mailfrom=*| eval
accountname=split(mailfrom,"@"), from_domain=mvindex(accountname,-1),
location=if(match(from_domain, "[^\n\r\s]+\.(com|net|org)"), "local",
"abroad") | stats count BY location
```

The first half of this search is similar to previous example. The `split()` function is used to break up the email address in the `mailfrom` field. The `mvindex` function defines the `from_domain` as the portion of the `mailfrom` field after the `@` symbol.

Then, the `if()` and `match()` functions are used.

- If the `from_domain` value ends with a `.com, .net., or .org`, the `location` field is assigned the value `local`.
- If `from_domain` does not match, `location` is assigned the value `abroad`.

The `eval` results are then piped into the `stats` command to count the number of results for each `location` value.

The results appear on the Statistics tab and look something like this:

location	count
abroad	3543
local	14136

Note: This example merely illustrates using the `match()` function. If you want to classify your events and quickly search for those events, the better approach is to use event types. Read more **about event types** in the *Knowledge manager manual*.

Defining calculated fields

If you find that you use a particular eval expression on a regular basis, consider defining the field as a calculated field. Doing this means that when you're writing a search, you can omit the eval expression and refer to the field like you do any other extracted field. When you run the search, the fields will be extracted at search time and will be added to the events that include the fields in the eval expressions.

Read more about how to configure this in "Define calculated fields" in the Knowledge Manager Manual.

Use lookup to add fields from lookup tables

You can match fields in your events to fields in external sources, such as lookup tables, and use these matches to add more information inline to your events.

A lookup table can be a static CSV file, a KV store collection, or the output of a Python script. You can also use the results of a search to populate the CSV file or KV store collection and then set that up as a lookup table. For more information about field lookups, see "Configure CSV and external lookups" and "Configure KV store lookups" in the *Knowledge Manager Manual*.

After you configure a fields lookup, you can invoke it from the Search app with the `lookup` command.

Example: Given a field lookup named `dnslookup`, referencing a Python script that performs a DNS and reverse DNS lookup and accepts either a host name or IP address as arguments -- you can use the lookup command to match the host name values in your events to the host name values in the table, and then add the corresponding IP address values to your events.

```
... | lookup dnslookup host OUTPUT ip
```

For a more extensive example using the Splunk script `external_lookup.py`, see "Reverse DNS Lookups for Host Entries" in the Splunk blogs.

Extract fields with search commands

You can use search commands to extract fields in different ways.

- The `rex` command performs field extractions using named groups in Perl regular expressions.
- The `extract` (or `kv`, for key/value) command explicitly extracts field and value pairs using default patterns.
- The `multikv` command extracts field and value pairs on multiline, tabular-formatted events.
- The `spath` command extracts field and value pairs on structured event data, such as XML and JSON.
- The `xmlkv` and `xpath` commands extract field and value pairs on XML-formatted event data.
- The `kvform` command extracts field and value pairs based on predefined form templates.

In Splunk Web, you can define field extractions on the **Settings > Fields > Field Extractions** page.

The following sections describe how to extract fields using regular expressions and commands. See About fields in the *Knowledge Manager Manual*.

Extract fields using regular expressions

The `rex` command performs field extractions using named groups in Perl regular expressions that you include in the search criteria. The `rex` command matches segments of your raw events with the regular expression and saves these matched values into a field.

In this example, values that occur after the strings `From:` and `To:` are saved into the **from** and **to** fields.

```
... | rex field=_raw "From: (?<from>.*) To: (?<to>.*)"
```

If a raw event contains `From: Susan To: Bob`, the search extracts the field name and value pairs: **from=***Susan* and **to=***Bob*.

For a primer on regular expression syntax and usage, see www.regular-expressions.info. The following are useful third-party tools for writing and testing regular expressions:

- regex101
- RegExr
- Debuggex

Extract fields from .conf files

The extract command forces field/value extraction on the result set. If you use the `extract` command without specifying any arguments, fields are extracted using field extraction stanzas that have been added to the `props.conf` file. You can also use the `extract` command to test field extractions that you add to the conf files.

Extract fields from events formatted as tables

Use the `multikv` command to force field and value extractions on multiline, tabular-formatted events. The `multikv` command creates a new event for each table row and derives field names from the table title.

Extract fields from events formatted in XML

The `xmlkv` command enables you to force field and value extractions on XML-formatted tags in event data, such as transactions from web pages.

Extract fields from XML and JSON documents

The `spath` command extracts information from structured data formats, such as XML and JSON, and store the extracted values in fields.

Extract fields from events based on form templates

The `kvform` command extracts field and value pairs from events based on form templates that are predefined and stored in `$SPLUNK_HOME/etc/system/local/`, or your own custom application directory in `$SPLUNK_HOME/etc/apps/`. For example, if `form=sales_order`, the search looks for a `sales_order.form`, and matches all processed events against that form to extract values.

If you have Splunk Cloud and want to use form templates for field extraction, file a Support ticket.

Evaluate and manipulate fields with multiple values

About multivalue fields

Multivalue fields are parsed at **search time**, which enables you to process the values in the search pipeline. Search commands that work with multivalue fields include makemv, mvcombine, mvexpand, and nomv. The eval and where commands support functions, such as `mvcount(), mvfilter(), mvindex(), and mvjoin()` that you can use with multivalue fields. See Evaluation functions in the *Search Reference* and the examples in this topic.

If you are using Splunk Enterprise, you can configure multivalue fields in the `fields.conf` file to specify how Splunk software detects more than one field value in a single extracted field value. Edit the `fields.conf` in `$SPLUNK_HOME/etc/system/local/`, or your own custom application directory in `$SPLUNK_HOME/etc/apps/`. For more information on how to do this, see Configure extractions of multivalue fields with fields.conf in the *Knowledge Manager Manual*.

If you are using Splunk Cloud and want to configure multivalue fields, file a Support ticket.

If your search produces results, such as a table, the results get written to the `results.csv.gz` file. The contents of the `results.csv.gz` file include fields that begin with "__mv_". These fields are for internal use only and are used to encode multivalue fields.

Evaluate multivalue fields

One of the more common examples of multivalue fields is email address fields, which typically appear two or three times in a single sendmail event--one time for the sender, another time for the list of recipients, and possibly a third time for the list of Cc addresses.

Count the number of values in a field

Use the `mvcount()` function to count the number of values in a single value or multivalue field.

In this example, mvcount() returns the number of email addresses in the To, From, and Cc fields and saves the addresses in the specified "_count" fields.

```
eventtype="sendmail" | eval To_count=mvcount(split(To,"@"))-1 | eval
From_count=mvcount(From) | eval Cc_count= mvcount(split(Cc,"@"))-1
```

This search takes the values in the `To` field and uses the `split` function to separate the email address on the @ symbol. The `split` function is also used on the `Cc` field for the same purpose.

If only a single email address exists in the `From` field, as you would expect, `mvcount(From)` returns 1. If there is no Cc address, the Cc field might not exist for the event. In that situation `mvcount(cc)` returns NULL.

Filter values from a multivalue field

Use the `mvfilter()` function to filter a multivalue field using an arbitrary Boolean expression. The `mvfilter` function works with only one field at a time.

In this example, mvfilter() keeps all of the values for the field `email` that end in `.net` or `.org`.

```
eventtype="sendmail" | eval email=mvfilter(match(email, "\.net$") OR
match(email, "\.org$"))
```

Note: This example also uses the `match()` function to compare the pattern defined in quotes to the value of `email`. See Evaluation functions in the *Search Reference*.

Return a subset of values from a multivalue field

Use the `mvindex()` function to reference a specific value or a subset of values in a multivalue field. Since the index numbering starts at 0, if you want to reference the 3rd value of a field, you would specify it as 2.

In this example, mvindex() returns the first email address in the "To" field for every email sent by Sender:

```
eventtype="sendmail" from=Sender@* | eval to_first=mvindex(to,0)
```

If you want to see the top 3 email addresses that Sender writes to, use the following search.

```
eventtype="sendmail" from=Sender@* | eval top_three=mvindex(to,0,2)
```

In this example, `top_three` is, itself, a multivalue field.

Manipulate multivalue fields

Use nomv to convert a multivalue field into a single value

You can use the `nomv` command to convert values of the specified multivalue field into one single value. The `nomv` command overrides the multivalue field configurations that are set in `fields.conf` file.

In this example for sendmail events, you want to combine the values of the senders field into a single value.

```
eventtype="sendmail" | nomv senders
```

Use makemv to separate a multivalue field

You can use the `makemv` command to separate multivalue fields into multiple single value fields. In this example for sendmail search results, you want to separate the values of the senders field into multiple field values.

```
eventtype="sendmail" | makemv delim="," senders
```

After you separate the field values, you can pipe it through other commands. For example, you can display the top senders.

```
eventtype="sendmail" | makemv delim="," senders | top senders
```

Use mvexpand to create multiple events based on a multivalue field

You can use the `mvexpand` command to expand the values of a multivalue field into separate events for each value of the multivalue field. In this example, new events are created for each value in the multivalue field, "foo".

```
... | mvexpand foo
```

Use mvcombine to create a multivalue field from similar events

Combine the values of "foo" with ":" delimiter.

```
... | mvcombine delim=":" foo
```

See also

Configure extractions of multivalue fields with fields.conf in the *Knowledge*

Manager Manual.

Calculate Statistics

About calculating statistics

This section discusses how to calculate summary statistics on events. When you think about calculating statistics with Splunk's search processing language (SPL), the `stats` command is probably what comes to mind first. The stats command generates reports that display summary statistics in a tabular format. Additionally, you can use the `chart` and `timechart` commands to create charted visualizations for summary statistics and the `geostats` command to create map visualizations for summary statistics of events that include geographical location fields.

The `stats`, `chart`, and `timechart` commands (and their related commands `eventstats`, `geostats` and `streamstats`) are designed to work in conjunction with statistical functions. For examples of searches using these commands and functions, read "Use the stats command and functions".

Later topics discuss how to:

- "Use stats with eval expressions and functions" to calculate statistics.
- "Add sparklines to report tables".

The Advanced statistics section contains topics on detecting anomalies, finding and removing outliers, detecting patterns, and time series forecasting.

Use the stats command and functions

This topic discusses how to use the statistical functions with the **transforming commands** `chart`, `timechart`, `stats`, `eventstats`, and `streamstats`.

- For more information about the stat command and syntax, see the "stats" command in the *Search Reference*.
- For the list of stats functions, see "Statistical and charting functions" in the *Search Reference*.

About the stats commands and functions

The `stats`, `streamstats`, and `eventstats` commands each enable you to calculate summary statistics on the results of a search or the events retrieved from an index. The `stats` command works on the search results as a whole. The `streamstats` command calculates statistics for each event at the time the event is seen, in a streaming manner. The `eventstats` command calculates statistics on all search results and adds the aggregation inline to each event for which it is relevant. See more about the differences between these commands in the next section.

The `chart command` returns your results in a data structure that supports visualization as a chart (such as a column, line, area, and pie chart). You can decide what field is tracked on the x-axis of the chart. The `timechart command` returns your results formatted as a time-series chart, where your data is plotted against an x-axis that is always a time field. Read more about visualization features and options in the Visualization Reference of the Data Visualization Manual.

The `stats`, `chart`, and `timechart` commands (and their related commands `eventstats` and `streamstats`) are designed to work in conjunction with statistical functions. The list of statistical functions lets you count the occurrence of a field and calculate sums, averages, ranges, and so on, of the field values.

For the list of statistical functions and how they're used, see "Statistical and charting functions" in the *Search Reference*.

Stats, eventstats, and streamstats

The `eventstats` and `streamstats` commands are variations on the `stats` command.

The `stats` command works on the search results as a whole and returns only the fields that you specify. For example, the following search returns a table with two columns (and 10 rows).

```
sourcetype=access_* | head 10 | stats sum(bytes) as ASumOfBytes by
clientip
```

The `ASumOfBytes` and `clientip` fields are the only fields that exist after the stats command. For example, the following search returns empty cells in the `bytes` column because it is not a result field.

```
sourcetype=access_* | head 10 | stats sum(bytes) as ASumOfBytes by
clientip | table bytes, ASumOfBytes, clientip
```

To see more fields other than `ASumOfBytes` and `clientip` in the results, you need to include them in the stats command. Also, if you want to perform calculations on any of the original fields in your raw events, you need to do that before the stats command.

The `eventstats` command computes the same statistics as the `stats` command, but it also aggregates the results to the original raw data. When you run the following search, it returns an events list instead of a results table, because the eventstats command does not change the raw data.

```
sourcetype=access_* | head 10 | eventstats sum(bytes) as ASumOfBytes by
clientip
```

You can use the `table` command to format the results as a table that displays the fields you want. Now, you can also view the values of `bytes` (or any of the original fields in your raw events) in your results.

```
sourcetype=access_* | head 10 | eventstats sum(bytes) as ASumOfBytes by
clientip | table bytes, ASumOfBytes, clientip
```

The `streamstats` command also aggregates the calculated statistics to the original raw event, but it does this at the time the event is seen. To demonstrate this, include the `_time` field in the earlier search and use `streamstats`.

```
sourcetype=access_* | head 10 | sort _time | streamstats sum(bytes) as
ASumOfBytes by clientip | table _time, clientip, bytes, ASumOfBytes
```

Instead of a total sum for each `clientip` (as returned by `stats` and `eventstats`), this search calculates a sum for each event based on the time that it is seen. The `streamstats` command is useful for reporting on events at a known time range.

Examples

Example 1

This example creates a chart of how many new users go online each hour of the day.

```
... | sort _time | streamstats dc(userid) as dcusers | delta dcusers as
deltadcusers | timechart sum(deltadcusers)
```

The `dc` (or `distinct_count`) function returns a count of the unique values of `userid` and renames the resulting field `dcusers`.

If you don't rename the function, for example "dc(userid) as dcusers", the resulting calculation is automatically saved to the function call, such as "dc(userid)".

The `delta` command is used to find the different between the current and previous dcusers value. Then, the sum of this delta is charted over time.

Example 2

This example calculates the median for a field, then charts the count of events where the field has a value less than the median.

```
... | eventstats median(bytes) as medbytes | eval
snap=if(bytes>=medbytes, bytes, "smaller") | timechart count by snap
```

Eventstats is used to calculate the median for all the values of bytes from the previous search.

Example 3

This example calculates the standard deviation and variance of calculated fields.

```
sourcetype=log4j ERROR earliest=-7d@d latest=@d | eval
warns=errorGroup+"-"+errorNum | stats count as Date_Warns_Count by
date_mday,warns | stats stdev(Date_Warns_Count), var(Date_Warns_Count)
by warns
```

This search returns errors from the last 7 days and creates the new field, warns, from extracted fields errorGroup and errorNum. The stats command is used twice. First, it calculates the daily count of warns for each day. Then, it calculates the standard deviation and variance of that count per warns.

Example 4

You can use the calculated fields as filter parameters for your search.

```
sourcetype=access_* | eval URILen = len(useragent) | eventstats
avg(URILen) as AvgURILen, stdev(URILen) as StdDevURILen| where URILen >
AvgURILen+(2*StdDevURILen) | chart count by URILen span=10 cont=true
```

In this example, eventstats is used to calculate the average and standard deviation of the URI lengths from `useragent`. Then, these numbers are used as filters for the retrieved events.

Use stats with eval expressions and functions

This topic discusses how to use eval expressions and functions within your stats calculation.

- For more information about the eval command and syntax, see the eval command in the *Search Reference*.
- For the list of eval functions, see Evaluation functions in the *Search Reference*.
- Also, you can read more about using the eval command to evaluate and manipulate fields in another section in this manual.

Example 1: Distinct counts of matching events

This example counts the IP addresses where the errors originate. This is similar to a search for events that is filtered for a specific error code, and then used with the stats command to count the IP addresses.

```
status=404 | stats dc(ip)
```

The best way to do this with an eval expression is:

```
status=404 | stats dc(eval(if(status=404, ip, NULL))) AS dc_ip
```

Example 2: Categorizing and counting fields

This example uses sample email data. You should be able to run this search on any email data by replacing the `sourcetype=cisco:esa` with the `sourcetype` value and the `mailfrom` field with email address field name in your data. For example, the email might be `To`, `From`, or `Cc`).

Find out how much of the email in your organization comes from .com, .net, .org or other top level domains.

The `eval` command in this search contains two expressions, separated by a comma.

```
sourcetype="cisco:esa" mailfrom=* | eval
accountname=split(mailfrom,"@"), from_domain=mvindex(accountname,-1) |
stats count(eval(match(from_domain, "[^\n\r\s]+\.com"))) AS ".com",
count(eval(match(from_domain, "[^\n\r\s]+\.net"))) AS ".net",
count(eval(match(from_domain, "[^\n\r\s]+\.org"))) AS ".org",
count(eval(NOT match(from_domain, "[^\n\r\s]+\.(com|net|org)"))) AS
"other"
```

- The first part of this search uses the `eval` command to break up the email address in the `mailfrom` field. The `from_domain` is defined as the portion of the `mailfrom` field after the `@` symbol.
    - The `split()` function is used to break the `mailfrom` field into a multivalue field called `accountname`. The first value of `accountname` is everything before the "@" symbol, and the second value is everything after.
    - The `mvindex()` function is used to set `from_domain` to the second value in the multivalue field `accountname`.
- The results are then piped into the `stats` command. The `count()` function is used to count the results of the `eval` expression.
- The`eval` uses the `match()` function to compare the `from_domain` to a regular expression that looks for the different suffixes in the domain. If the value of `from_domain` matches the regular expression, the `count` is updated for each suffix, `.com`, `.net`, and `.org`. Other domain suffixes are counted as `other`.

The results appear on the Statistics tab and look something like this:

.com	.net	.org	other
4246	9890	0	3543

Add sparklines to search results

If you are working with `stats` and `chart` searches, you can increase their usefulness and overall information density by adding sparklines to their result tables. Sparklines are inline charts that appear within table cells in search results, and are designed to display time-based trends associated with the primary key of each row.

For example, say you have this search, set to run over events from the **Last 15 minutes**:

```
index=_internal | chart count by sourcetype
```

This search returns a two-column results table that shows event counts for the source types that have been indexed to `_internal` in the last 15 minutes. The first column lists each `sourcetype` found in the past hour's set of `_internal` index events; this is the primary key for the table. The second column, `count`, displays the event counts for each listed source type:

You can add sparklines to the results of this search by adding the `sparkline` function to the search itself:

```
index=_internal | chart sparkline count by sourcetype
```

This results in a table that is almost the same as the preceding one, except that now, for each row you have a sparkline chart that shows the event count trend for each listed source type over the past 15 minutes.

Now you can easily see patterns in your data that may have been invisible before. Some search activity apparently caused a bump in most `index=_internal` source types about three quarters into the 15 minute span. And `splunkd` has what almost looks like a regular heartbeat running over the entire span of time.

Each sparkline in a table displays information in relation to the other events represented in that sparkline, but not in relation to the other sparklines. A peak in one sparkline does not necessarily have the same value as a peak in another sparkline.

Using sparklines with the stats and chart commands

You always use the sparklines feature in conjunction with `chart` and `stats` searches, because it is a function of those two search commands. It is not a command by itself. The functionality of sparklines is the same for both search commands.

Sparklines are not available as a dashboard chart visualization by themselves, but you can set up a dashboard panel with a table visualization that displays sparklines. For more information, see the "Visualization reference" topic in the Splunk Data Visualizations Manual.

For more information about the `chart` and `stats` commands, including details on the syntax around the `sparkline` function, see chart and stats in the *Search Reference*.

Example: Stats, sparklines, and earthquake data

Here are some examples of `stats` searches that use sparklines to provide additional information about earthquake data.

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the **USGS Earthquake Feeds** and add it as an input.

Let's say you want to use the USGS Earthquakes data to show the locations that had the most earthquakes over the past month, a column that shows the average quake magnitude for each location. You could use the following search:

```
source="all_month.csv" | stats sparkline count, avg(mag) by
locationSource | sort count
```

This search returns the following table, with sparklines that illustrate the quake count over the course of the month for each of the top earthquake locations:

Right away you can see differences in quake distribution between the different locations.

You can click on a sourceLocation to see the actual events that are included in the location calculations.

For the **avg(mag)** column, you can use the Format icon to change the number formatting in that column.

You can easily get the minimum and maximum count for a particular region by mousing over the sparkline; in this example you can see that in Southern Alaska, the minimum count of quakes experienced in a single day during the 7-day period was 1, while the maximum count per day was 6.

But what if you want your sparkline to represent not only the earthquake count, but also the relative average magnitude of the quakes affecting each region? In other words, how can you make the sparkline line chart represent average quake magnitude for each "time bucket" (segment) of the chart?

Try a search like this:

```
source="all_month.csv"| stats sparkline(avg(mag),6h) as magnitude_trend
count, avg(mag) by locationSource | sort - count
```

This search produces a sparkline for each region that shows the average quake magnitude for the quake events that fall into each segment of the sparkline. By specifying a dash (-) after **sort**, the results are sorted in descending order.

But it does a bit more than that. It also asks that the sparkline divide itself up into smaller chunks of time. The preceding table had a sparkline that was divided up by day, so each data point in the sparkline represented an event count for a full 24 hour period. This is why those sparklines were so short.

The addition of the `6h` to the search language overrides this default and displays sparklines that are broken up into discrete six-hour chunks, which makes it easier to see the distribution of events along the sparkline for the chosen time range.

The search also renames the sparkline column as **magnitude_trend** to make it easier to understand.

Now you can see that the quakes for the **tul** location are spread out more evenly than the previous search suggested.

Advanced Statistics

About advanced statistics

The previous section shows you how to calculate basic statistics using `stats`, `eval`, and how to create sparkline charts.

In this section we discuss how to detect anomalies in your data. This could include finding outliers to identify anomalies or spikes in your data. You might want to remove outliers that unnecessarily skew your calculations or the way your charts plot the data. You can detect patterns in your data, grouping events based on how similar the events are to each other. If there are patterns and correlations to events that you monitor, you can use them to predict future activity. With this knowledge, you can proactively send alerts based on thresholds and perform "what-if" analyses to compare various scenarios. All of this and more is possible with advanced statistics.

- Commands for advanced statistics
- About anomaly detection
- Finding and removing outliers
- Detecting anomalies
- Detecting patterns
- About time series forecasting
- Machine Learning Toolkit

See also

Use the stats command and functions

Use stats with eval expressions and functions

Add sparklines to report tables

Commands for advanced statistics

Now it is time to learn about the rich set of commands that you can use to perform advanced statistics. The following tables organize these commands by category.

Commands that find anomalies

These commands are used to find anomalies in your data. You can search for uncommon or outlying events and fields, or cluster similar events together.

Command	Description
`analyzefields`	Analyzes numeric fields for their ability to predict another discrete field.
`anomalies`	Computes an "unexpectedness" score for an event.
`anomalousvalue`	Finds and summarizes irregular, or uncommon, search results.
`anomalydetection`	Computes a probability for each event and detects unusually small probabilities.
`cluster`	Groups similar events together.
`kmeans`	Partitions the events into k clusters, with each cluster defined by its mean value. Each event belongs to the cluster with the nearest mean value.
`outlier`	Removes outlying numerical values.
`rare`	Displays the least common values of a field.

Commands for predicting and trends

These commands predict future values and calculate trendlines that can be used to create visualizations.

Command	Description
`predict`	Predicts future values of a time series.
`trendline`	Computes moving averages of fields.
`x11`	Looks for trends in a time series by removing repeating patterns.

Commands for reports, charts, and maps

These commands are used to build **transforming searches**. These commands return statistical data tables that are required for charts and other kinds of data visualizations.

Command	Description

`addtotals`	Computes the sum of all numeric fields for each search result.
`contingency`	Builds a contingency table, a co-occurrence matrix, for the values of two fields.
`correlate`	Calculates the correlation between different fields.
`eval`	Calculates mathematical, string, or boolean expressions. Puts the value into a new field.
`eventstats`	Adds summary statistics to all search results.
`geostats`	Generates statistics to display geographic data and summarize the data on maps.
`outlier`	Removes outlying numerical values.
`rare`	Displays the least common values of a field.
`stats`	Calculates aggregate statistics over the results set, such as average, count, and sum.
`streamstats`	Adds summary statistics about the preceding events to each search result.
`timechart`	Create a time series chart and corresponding table of statistics. See "Statistical and charting functions" in the *Search Reference*.
`trendline`	Computes moving averages of fields.

About anomaly detection

This section describes anomaly detection. For a complete list of topics on detecting anomalies, finding and removing outliers, detecting patterns, and time series forecasting see About advanced statistics, in this manual.

Overview of anomaly detection

An anomaly is a deviation from the expected behavior of the system. An anomaly can be:

- A single event
- A sequence of events
- A sequence of transactions
- Complex patterns

Examples of common use cases for anomaly detection include:

Industry	Use case example
IT	Identifying a distributed denial of service (DDoS) attack from IP address ranges.
Marketing	Rare but high-value customer purchase patterns.
Product	Rare or previously unknown method of using a product that yields better results or yields results more efficiently than known methods.
Security	Faster-than-human transactions. Detecting when transactions are being performed much more quickly by one user than by others. This could indicate a bot or an attempt to probe security measures.

Effective anomaly detection

To perform effective anomaly detection, put all of the data in one place. If you do not have your machine and business data in the same place, you cannot perform a comprehensive analysis.

Begin tracking IT and business performance metrics. Additionally, create a baseline data image which shows the current state of your system.

See also

About advanced statistics, Finding and removing outliers, Detecting anomalies

Finding and removing outliers

This section describes outliers. For a complete list of topics on detecting anomalies, detecting patterns, and time series forecasting see About advanced statistics, in this manual.

What is an outlier?

An outlier is a data point that is far removed from the typical distribution of data points.

In some situations you might want to simply identify the outliers. In other situations you might want to remove the outliers so that they do not skew your statistical results or create issues displaying data in your charts.

Statistical outliers

A *statistical outlier* is a data point that is far removed from some measure of centrality. Typical measures of centrality are mean, median, and mode. Mean is the average value. Median is the value at the middle of a sorted list of all values. Mode is the most frequent value.

Centrality measure	Splunk centrality commands
Mean	`stats avg(`*`field`*`)`
Median	`stats median(`*`field`*`)`
Mode	`stats mod(`*`field`*`)`

There are different types of statistical outliers. Outliers can be far from the mean, far from the median, or a small number relative to the mode. An outlier can also be a data point that is far removed from the typical range of data points.

Identifying outliers

There are several methods you can use to identify outliers. Often these methods involve calculating some measure of centrality and then identifying the outliers.

In some cases outliers are identified when you notice an anomaly. For example, when you chart the data and you notice that the axis is skewed. An outlier can be the culprit.

Use the number of statistical outliers as a bellwether. If you see more statistical outliers than usual, that phenomenon itself is an anomaly.

To calculate the centrality measure, you can use the following commands.

Centrality measure	Splunk centrality commands

Standard deviation	`stats stdev(`*`field`*`)`
Quartiles and percentiles	`stats perc75(`*`field`*`) perc25(`*`field`*`)`
Top 3 most frequent values	`top 3 `*`field`*

In many cases, you need additional commands to calculate the information that you are looking for. The following sections contain some of these examples.

Calculate the lower and upper boundaries of an acceptable range to identify outliers

The following example takes the first 500 events from the `quote.csv` file. The `streamstats` command and a moving window of 100 events are used to calculate the average and standard deviation. The average and standard deviation are used with the `eval` command to calculate the lower and upper boundaries. A sensitivity is added into the calculation by multiplying the `stdev` by 2. The `eval` command uses those boundaries to identify the outliers. The outliers are then sorted in descending order.

```
| inputlookup quote.csv | head 500 | eval _time=(round(strptime(time,
"%Y-%m-%d %H:%M:%SZ"))) | streamstats window=100 avg("price") as avg
stdev("price") as stdev | eval lowerBound=(avg-stdev*2) | eval
upperBound=(avg+stdev*2) | eval isOutlier=if('price' < lowerBound OR
'price' > upperBound, 1, 0) | fields "_time", "symbol", "sourcetype",
"time", "price", "lowerBound", "upperBound", "isOutlier" | sort -
isOutlier
```

Use the interquartile range (IQR) to identify outliers

The following example takes the first 500 events from the `quote.csv` file. The `eventstats` command is used to calculate the median, the 25th percentile (p25), and the 75th percentile (p75). The IQR is calculated with the `eval` command by subtracting the percentiles. The median and the IQR are used with the `eval` command to calculate the lower and upper boundaries. A sensitivity is added into the calculation by multiplying the IQR by 20.The `eval` command uses those boundaries to identify the outliers. The outliers are then sorted in descending order.

```
| inputlookup quote.csv | head 500 | eval _time=(round(strptime(time,
"%Y-%m-%d %H:%M:%SZ"))) | eventstats median("price") as median
p25("price") as p25 p75("price") as p75 | eval IQR=(p75-p25) | eval
lowerBound=(median-IQR*20) | eval upperBound=(median+IQR*20) | eval
isOutlier=if('price' < lowerBound OR 'price' > upperBound, 1, 0) |
fields "_time", "symbol", "sourcetype", "time", "price", "lowerBound",
"upperBound", "isOutlier" | sort - isOutlier
```

Removing outliers in charts

You can use the outlier command to remove outlying numerical values from your search results.

You have the option to remove or transform the events with outliers. The `remove` option removes the events. The `transform` option truncates the outlying value to the threshold for outliers. The threshold is specified with the `param` option.

A value is considered an outlier if the value is outside of the `param` threshold multiplied by the inter-quartile range (IQR). The default value for `param` is 2.5.

Create a chart of web server events, transform the outlying values

For a timechart of web server events, transform the outlying average CPU values.

```
host=''web_server'' 404 | timechart avg(cpu_seconds) by host | outlier
action=transform
```

Remove outliers that interfere with displaying the y-axis in a chart

Sometimes when you create a chart, a small number of values are so far from the other values that the chart is rendered unreadable. You can remove the outliers so that the chart values are visible.

```
index=_internal source=*access* | timechart span=1h max(bytes) |
fillnull | outlier
```

Remove outliers using the three-sigma rule across transactions

This example uses the `eventstats` command to calculate the average and the standard deviation. The three-sigma limit is then calculated. The `where` command filters search results. Only the events with the duration less than the three-sigma limit are returned.

```
... | eval durationMins = (duration/60) | eventstats avg(durationMins)
as Avrg, stdev(durationMins) as StDev | eval threeSigmaLimit = (Avrg +
(StDev * 3)) | where durationMins < threeSigmaLimit
```

Manage alerts for outliers

When setting up an alert, it is important to review the outlier threshold you have set.

* If the threshold value is too low, too many alerts are returned for non-critical outliers
* If the threshold value is too high, not enough alerts are returned and you might not identify the outliers

Typically a small percentage of events in your data are outliers. If you have 1,000,000 events a day and 5% of the events are outliers, setting an alert would trigger 50,000 alert actions unless you specify a throttle. A *throttle* suppresses additional alerts that have the same field value in a given time range.

For example, your search returns on average 100 events every minute. You only want to be alerted when the status for an event is 404. You can setup the alert to perform the alert action one time every 60 seconds instead of alerting you for every event that has a 404 status in that 60 second window.

Set alert throttling

You set throttling as part of setting an alert.

1. Determine what percent of your events are outliers.

2. Under **Settings**, choose **Searches, reports, and alerts**.

3. Under Schedule and alert, click the **Schedule this search** check box. The screen expands to display the scheduling and alerting options.

4. Specify the alert condition and mode.

5. Mark the **Throttling** check box and specify when the throttling expires.

6. Specify the alert action.

7. Click **Save**.

See also

About advanced statistics
SPL commands: outlier, stats, top

Detecting anomalies

Finding spikes in your data

You want to identify spikes in your data. Spikes can show you where you have peaks (or troughs) that indicate that some metric is rising or falling sharply. There are all sorts of spikes. Traffic spikes, sales spikes, spikes in the number of returns, spikes in database load. Whatever type of spike you are interested in, you want to watch for it and then perhaps take some action to address those spikes.

You can use a moving trendline to help you see the spikes. Run a search followed by the trendline command using a field that you want to create a trendline for.

For example, on web access data, you could chart an average of the `bytes` field.

```
sourcetype=access* | timechart avg(bytes) as avg_bytes
```

To add another line or bar series to the chart for the simple moving average (sma) of the last 5 values of `bytes`, use this command:

```
... | trendline sma5(avg_bytes) as moving_avg_bytes
```

If you want to clearly identify spikes, you might add an additional series for spikes. The following search adds a field called "spike" that indicates when the average number of bytes exceeds twice the moving average.

```
... | eval spike=if(avg_bytes > 2 * moving_avg_bytes, 10000, 0)
```

The 10000 here is arbitrary and you should choose a value relevant to your data that makes the spike noticeable. Changing the formatting of the y-axis to Log scale also helps.

Putting this all together, the search is:

```
sourcetype=access* | timechart avg(bytes) as avg_bytes | trendline
sma5(avg_bytes) as moving_avg_bytes | eval spike=if(avg_bytes > 2 *
moving_avg_bytes, 10000, 0)
```

This search uses a simple moving average for the last 5 results (sma5). Explore with different simple moving average values to determine the best simple moving average to use to identify the spikes.

The trendline command also supports the exponential moving average (ema) and the weighted moving average (wma).

Alternatively, you can bypass the charting altogether and replace the `eval` command with the `where` command to filter your results.

```
... | where avg_bytes > 2 * moving_avg_bytes
```

And by looking at the table view or setting an alert, you will see when the `avg_bytes` spiked.

See also

About advanced statistics, eval, trendline

Detecting patterns

This section describes detecting patterns in your data. For a complete list of topics on detecting anomalies, finding and removing outliers, and time series forecasting see About advanced statistics, in this manual.

Detecting patterns in events

The cluster command is a powerful command for detecting patterns in your events. The command groups events based on how similar they are to each other. The `cluster` command groups events based on the contents of the `_raw` field, unless you specify another field.

When you use the `cluster` command, two new fields are appended to each event.

- The `cluster_count` is the number of events that are part of the cluster. This is the cluster size.
- The `cluster_label` specifies which cluster the event belongs to. For example, if the search returns 10 clusters, then the clusters are labeled from 1 to 10.

Anomalies come in small or large groups (or clusters) of events. A small group might consist of 1 or 2 login events from a user. An example of a large group of events might be a DDoS attack of thousands of similar events.

Use the cluster command parameters wisely

- Use the `labelonly=true` parameter to return all of the events. If you use `labelonly=false`, which is the default, then only one event from each cluster is returned.
- Use the `showcount=true` parameter so that a `cluster_count` field is added to all of the events. If `showcount=false`, which is the default, the event count is not added to the event.
- The threshold parameter `t` adjusts the cluster sensitivity. The smaller the threshold value, the fewer the number of clusters.

Other commands to use with the cluster command

- Use the dedup command on the cluster_label column to see the most recent grouped events within each cluster.
- To group the events and make the results more readable, use the sort command with the cluster columns. Sort the cluster_count column based on the number of clusters.
    - For small groups of events, sort the `cluster_count` column in ascending order.
    - For large groups of events sort the `cluster_count` column in descending order.
    - Sort the `cluster_label` column in ascending order. Cluster labels are numeric. Sorting in ascending order organizes the events by label, in numerical order.

Return the 3 most recent events in each cluster

The following search uses the CustomerID in the `sales_entries.log` file. Setting `showcount=true` ensures that all events get a `cluster_count`. The cluster threshold is set to 0.7. Setting `labelonly=true` returns the incoming events. The `dedup` command is used to see the 3 most recent events within each cluster. The results are sorted in descending order to group the events.

```
source="/opt/log/ecommsv1/sales_entries.log" CustomerID | cluster
showcount=true t=0.7 labelonly=true | table _time, cluster_count,
cluster_label, _raw | dedup 3 cluster_label | sort -cluster_count,
cluster_label, - _time
```

If you do not set `labelonly=true`, then only one event from each cluster is returned.

See also

About advanced statistics, dedup, cluster, sort

About time series forecasting

You can forecast time series data in a number of ways. For example:

- Capacity planning to determine hardware requirements for virtual environments and forecast energy consumption
- Forecast earnings and other business metrics
- Enhanced monitoring of key components which can detect system failures and prevent outages before they occur

You can use reports and dashboards to monitor activity as it is happening, then drill down into events and do a root-cause analysis to learn why something happened. If there are patterns and correlations in events that you monitor, you can use them to predict future activity. With this knowledge, you can proactively send alerts based on thresholds and perform "what-if" analyses to compare various scenarios.

Commands for time series forecasting

The Splunk search language includes two forecasting commands: predict and x11.

- The predict command enables you to use different forecasting algorithms to predict future values of single and multivalue fields.
- The x11 command, which is named after the X11 algorithm, removes seasonal fluctuations in fields to expose the real trend in your underlying data series.

Forecasting algorithms

You can select from the following algorithms with the `predict` command: LL, LLP, LLT, LLB, and LLP5. Each of these algorithms are variations of the Kalman filter.

Algorithm option	Algorithm name	Description

LL	Local level	This is a univariate model with no trends and no seasonality. Requires a minimum of 2 data points.
LLP	Seasonal local level	This is a univariate model with seasonality. The periodicity of the time series is automatically computed. Requires the minimum number of data points to be twice the period.
LLT	Local level trend	This is a univariate model with trend but no seasonality. Requires a minimum of 3 data points.
LLB	Bivariate local level	This is a bivariate model with no trends and no seasonality. Requires a minimum of 2 data points. LLB uses one set of data to make predictions for another. For example, assume it uses dataset Y to make predictions for dataset X. If the holdback=10, the LLB algorithm uses the last 10 data points of Y to make predictions for the last 10 data points of X.
LLP5		Combines LLT and LLP models for its prediction.

For more information, see the "predict command" in the *Search Reference*.

Forecasting seasonality with the x11 command

The seasonal component of your time-series data is either additive or multiplicative, which is reflected in the two types of seasonality that you can calculate with the x11 command: `add()` for additive and `mult()` for multiplicative.

How do you know which type of seasonality to adjust from your data? The best way to describe the difference between an additive and a multiplicative seasonal component is with an example: The annual sales of flowers will peak on and around certain days of the year, such as Valentine's Day and Mother's Day.

During Valentine's Day, the sale of roses might increase by X dollars every year. This dollar amount is independent of the normal level of the series, and you can add X dollars to your forecasts for Valentine's Day every year, making this time series a candidate for an additive seasonal adjustment. In an additive seasonal adjustment, each value of a time series is adjusted by adding or subtracting a quantity that represents the absolute amount by which that value differs from normal in that season.

Alternatively, in a multiplicative seasonal component, the seasonal effect expresses itself in percentage terms. The absolute magnitude of the seasonal variations increases as the series grows over time. For example, the number of roses sold during Valentine's Day might increase by 40% or a factor of 1.4. When

the sales of roses is generally weak, the absolute (dollar) increase in Valentine's Day sales will also be relatively weak. However, the percentage will be constant. And, if the sales of roses are strong, then the absolute (dollar) increase will be proportionately greater. In a multiplicative seasonal adjustment, this pattern is removed by dividing each value of the time series by a quantity that represents the percentage from normal or divided by a factor that is typically observed in that season.

When plotted on a chart, these two types of seasonal components show distinguishing characteristics:

- The additive seasonal series shows steady seasonal fluctuations, regardless of the overall level of the series.
- The multiplicative seasonal series shows varying size of seasonal fluctuations that depend on the overall level of the series.

For more information, see the "x11 command" in the *Search Reference*.

See also

About advanced statistics, predict, x11

Machine Learning

This section describes the Splunk Machine Learning Toolkit. For a complete list of topics on detecting anomalies, finding and removing outliers, detecting patterns, and time series forecasting see About advanced statistics.

Splunk Machine Learning Toolkit

The Splunk Machine Learning Toolkit app delivers new SPL commands, custom visualizations, assistants, and examples to explore a variety of machine learning concepts.

Each assistant includes end-to-end examples with datasets, plus the ability to apply the visualizations and SPL commands to your own data. You can inspect the assistant panels and underlying code to see how it all works. See the Splunk Machine Learning Toolkit User Guide for information.

Group and Correlate Events

About event grouping and correlation

Event correlation is finding relationships between seemingly unrelated events in data from multiple sources to answer questions like, "how far apart in time did a specific set of events occur?" or "what's the total amount of time it took for a transaction to complete?"

Splunk software supports event correlations using time and geographic location, transactions, sub-searches, field lookups, and joins.

- Identify relationships based on the time proximity or geographic location of the events. Use this correlation in any security or operations investigation, where you might need to see all or any subset of events that take place over a given time period or location.
- Track a series of related events, which may come from separate IT systems and data sources, together as a single transaction. Identify the amount of time it took to complete the transaction and the number of events within a single transaction.
- Use a sub-search to take the results of one search and use them in another. Create conditional searches, where you see the results of a search only if the sub-search meets certain thresholds.
- Correlate your data to external sources with lookups.
- Use SQL-like inner and outer joins to link two completely different data sets together based on one or more common fields.

This chapter discusses three methods for correlating or grouping events:

- Use time to identify relations between events
- Use subsearch to correlate events
- Use transactions to identify and group related events

You can also use field lookups and other features of the search language. Depending on your search criteria and how you want to define your groupings, you may be able to use a search command, such as append, associate, contingency, join, or stats. Sometimes, there is no single command that you can use.

If you're not sure where to start, the following flow chart can help you decide whether to use a lookup, define a transaction, or try another search command to

define your event grouping.

In most cases, you can accomplish more with the stats command or the transaction command; and these are recommended over using the `join` and `append` commands. You can read more about when to use `stats` and `transaction` in the topic "About transactions" later in this chapter. You can also read more about the stats commands in the "Calculate Statistics" chapter of this manual.

Note: The information for this diagram was provided by Nick Mealy.

Use time to identify relationships between events

Time is crucial for determining what went wrong ? you often know when. Splunk software enables you to identify baseline patterns or trends in your events and compare it against current activity.

You can run a series of time-based searches to investigate and identify abnormal activity and then use the timeline to drill into specific time periods. Looking at events that happened around the same time can help correlate results and find the root cause.

Read more about how to "Use the timeline to investigate events" in this manual.

About transactions

A **transaction** is any group of conceptually-related events that spans time, such as a series of events related to the online reservation of a hotel room by a single customer, or a set of events related to a firewall intrusion incident. A **transaction type** is a configured transaction, saved as a field and used in conjunction with the `transaction` command. Any number of data sources can generate transactions over multiple log entries.

Transaction search

A transaction search is useful for a single observation of any physical event stretching over multiple logged events. Use the transaction command to define a transaction or override transaction options specified in `transactiontypes.conf`.

One common use of a transaction search is to group multiple events into a single meta-event that represents a single physical event. For example, an **out of memory problem** could trigger several database events to be logged, and they can all be grouped together into a transaction.

To learn more, see Identify and group events into transactions in this manual.

Using stats instead of transaction

Both the stats command and the transaction command are similar in that they enable you to aggregate individual events together based on field values.

The `stats` command is meant to calculate statistics on events grouped by one or more fields and discard the events (unless you are using eventstats or streamstats). On the other hand, except for the duration between first and last events and the count of events, the `transaction` command does not compute statistics over the grouped events. Additionally, it retains the raw event and other field values from the original event and enables you to group events using much more complex criteria, such as limiting the grouping by time span or delays and requiring terms to define the start or end of a group.

The `transaction` command is most useful in two specific cases:

1. When a unique ID (from one or more fields) alone is not sufficient to discriminate between two transactions. This is the case when the identifier is reused, for example web sessions identified by cookie or client IP. In this case, time spans or pauses are also used to segment the data into transactions. In

other cases, when an identifier is reused, for example in DHCP logs, a particular message may identify the beginning or end of a transaction.

2. When it is desirable to see the raw text of the events combined rather than an analysis on the constituent fields of the events.

In other cases, it's usually better to use the `stats` command, which performs more efficiently, especially in a distributed environment. Often there is a unique ID in the events and `stats` can be used.

For example, to compute the statistics on the duration of trades identified by the unique ID `trade_id`, the following searches will yield the same answer:

```
... | transaction trade_id | chart count by duration span=log2
```

and

```
... | stats range(_time) as duration by trade_id | chart count by
duration span=log2
```

If however, the `trade_id` values are reused but each trade ends with some text, such as "END", the only solution is to use this `transaction` search:

```
... | transaction trade_id endswith=END | chart count by duration
span=log2
```

On the other hand, if `trade_id` values are reused, but not within a 10 minute duration, the solution is to use the following `transaction` search:

```
... | transaction trade_id maxpause=10m | chart count by duration
span=log2
```

Read more about "About event grouping and correlation" in an earlier chapter in this manual.

Transactions and macro search

Transactions and macro searches are a powerful combination that allow substitution into your transaction searches. Make a transaction search and then save it with `$field$` to allow substitution.

For an example of how to use macro searches and transactions, see Define and use search macros in the *Knowledge Manager Manual*.

Identify and group events into transactions

You can search for related events and group them into one single event, called a *transaction* (sometimes referred to as a session).

Transactions can include:

- Different events from the same source and the same host.
- Different events from different sources from the same host.
- Similar events from different hosts and different sources.

Search for transactions using the `transaction` command either in Splunk Web or at the CLI. The `transaction` command yields groupings of events which can be used in reports. To use `transaction`, either call a transaction type (that you configured via transactiontypes.conf), or define transaction constraints in your search by setting the search options of the `transaction` command.

Transaction search options

Transactions returned at search time consist of the raw text of each event, the shared event types, and the field values. Transactions also have additional data that is stored in the fields: `duration` and `transactiontype`.

- **`duration`** contains the duration of the transaction (the difference between the timestamps of the first and last events of the transaction).
- **`transactiontype`** is the name of the transaction (as defined in `transactiontypes.conf` by the transaction's stanza name).

You can add `transaction` to any search. For best search performance, craft your search and then pipe it to the transaction command. For more information see the topic on the `transaction` command in the Search Reference manual.

Follow the `transaction` command with the following options. **Note:** Some `transaction` options do not work in conjunction with others.

`name=<transaction-name>`

- Specifies the name of a stanza from `transactiontypes.conf`. Use this to invoke a **transaction type** that you have already configured for reuse. If other arguments are provided, they overule values specified for the same arguments in the transaction rule. For example, if `web_purchase`, the transaction rule you're invoking, is configured with `maxevents=10`, but you'd

like to run it with a different value for `maxevents`, add `maxevents` to the search string with the value you want:

```
sourcetype=access_* | transaction name=web_purchase maxevents=5
```

[field-list]

- This is a comma-separated list of fields, such as `... | transaction host,cookie`
- If set, each event must have the same field(s) to be considered part of the same transaction.
- Events with common field names and different values will not be grouped.
  - For example, if you add `... | transaction host`, then a search result that has `host=mylaptop` can never be in the same transaction as a search result with `host=myserver`.
  - A search result that has no `host` value can be in a transaction with a result that has `host=mylaptop`.

`match=closest`

- Specify the matching type to use with a transaction definition.
- The only value supported currently is closest.

`maxspan=[<integer>s | m | h | d]`

- Set the maximum duration of one transaction.
- Can be in seconds, minutes, hours or days.
  - For example: 5s, 6m, 12h or 30d.
- Defaults to `maxspan=-1`, for an "all time" timerange.

`maxpause=[<integer> s|m|h|d]`

- Specifies the maximum pause between transactions.
- Requires there be no pause between the events within the transaction greater than maxpause.
- If the value is negative, the maxspause constraint is disabled.
- Defaults to `maxpause=-1`.

`startswith=<string>`

- A search or eval-filtering expression which, if satisfied by an event, marks the beginning of a new transaction.
- For example:
  - `startswith="login"`
  - `startswith=(username=foobar)`
  - `startswith=eval(speed_field < max_speed_field)`
  - `startswith=eval(speed_field < max_speed_field/12)`
- Defaults to `""`.

`endswith=<transam-filter-string>`

- A search or eval-filtering expression which, if satisfied by an event, marks the end of a transaction.
- For example:
  - `endswith="logout"`
  - `endswith=(username=foobar)`
  - `endswith=eval(speed_field < max_speed_field)`
  - `endswith=eval(speed_field < max_speed_field/12)`
- Defaults to `""`.

For `startswith` and `endswith`, `<transam-filter-string>` is defined with the following syntax: `"<search-expression>" | (<quoted-search-expression>) | eval(<eval-expression>`

- `<search-expression>` is a valid search expression that does not contain quotes.
- `<quoted-search-expression>` is a valid search expression that contains quotes.
- `<eval-expression>` is a valid eval expression that evaluates to a boolean.

Examples:

- search expression: `(name="foo bar")`
- search expression: `"user=mildred"`
- search expression: `("search literal")`
- eval bool expression: `eval(distance/time < max_speed)`

Example transaction search

Run a search that groups together all of the web pages a single user (or client IP address) looked at over a time range.

This search takes events from the access logs, and creates a transaction from events that share the same `clientip` value that occurred within 5 minutes of each other (within a 3 hour time span).

```
sourcetype=access_combined | transaction clientip maxpause=5m
maxspan=3h
```

Refer to the transaction command topic in the Search Reference Manual for more examples.

Manage Jobs

About jobs and job management

Each time you run a search, create a pivot, open a report, or load a dashboard panel, the Splunk software creates a **job** in the system. When you run a search, you are creating an **ad hoc search**. Pivots, reports, and panels are powered by **saved searches**.

A job is a process that tracks information about the ad hoc search or saved search. The information that is tracked includes the owner of the job, the app that the job was run on, how many events were returned, and how long the job took to run.

Each job process creates a **search artifact**. The artifact contains the results and associated metadata that are returned at the time that the ad hoc search or saved search was run.

Inspecting jobs and managing jobs

There are several ways that you can look at information about your jobs. You can inspect a job or you can manage a job.

Search Job Inspector
: Use the Search Job Inspector to view information about the current job, such as job execution costs and search job properties. See View search job properties.

Jobs manager page

Use the Jobs manager page to view information about recent jobs. If you have the Admin role or a role with an equivalent set of capabilities, you can manage the search jobs run by other users. See Manage search jobs.

Job menu

After you run a search or open a report in Splunk Web, you can access and manage information about the search job without leaving the Search page. While the search is running, paused, or finalized, click **Job** and choose from the available options.

- **Edit the job settings.** Select this to open the Job Settings dialog, where you can change the job read permissions, extend the job lifetime, and get a URL for the job. You can use the URL to share the job with others, or to create a bookmark to the job from your web browser.
- **Send the job to the background.** Select this if the search job is slow to complete and you want to you work on other Splunk activities, including running a new search job. The job continues to run in the background.
- **Inspect the job.** Opens a separate window and displays information and metrics for the search job using the Search Job Inspector.
- **Delete the job.** Use this to delete a job that is currently running, is paused, or which has finalized. After you delete the job, you can still save the search as a report.

Edit search job settings

You can open the Job Settings dialog when a search job is running, paused, or finalized. Just click **Job** and select **Edit Job Settings**.

Sharing jobs

There are several ways to share a job with other Splunk users. You can change the job permissions or send a link to the job. This can be handy if you want another user to see the results returned by the job. See Sharing and exporting jobs.

Job lifetimes

When you run a new search, a job is retained in the system for a period of time, called the job **lifetime**. The default lifetime is 10 minutes. The lifetime starts from the moment the job is run. See Extending job lifetimes in this manual.

Managing long-running jobs

Sometimes a search job runs for a long time. You might want to edit the search to change the search criteria, or you might want to pause the search or run the search in the background.

Autopause long-running jobs

To handle inadvertently long-running search jobs, you can autopause a job. This feature is enabled by default only for summary dashboard clicks, to deal with the situation where a user mistakenly initiates "all time" searches.

When autopause is enabled for a particular search view, the search view includes an autopause countdown field during the search. If the search time limit has been reached, an information window will appear to inform the user that the search has been paused. It offers the user the option of resuming or finalizing the search. By default, the limit before autopause is 30 seconds.

Managing jobs when a computer goes into sleep mode

When a search is run in Splunk Web from a computer that is not a Splunk server and the computer changes to sleep or hibernate mode, the underlying search process is stopped. The Splunk software interprets the change to sleep or hibernate mode as if the browser tab in which the software is running has been closed and is no longer being used.

To avoid this issue, use one of the following techniques:

- Send the job to the background. The job continues to run in the background even when your computer goes into sleep or hibernate mode. From the **Job** menu, select **Send Job to Background**.
- Save and schedule the search. The search runs independently from the computer that was used to create the search. You will need to decide if you want to save and schedule the search as a report, dashboard or an alert. See Saving searches and Scheduling searches.
- Share the job. The lifetime of the job is automatically extended to 7 days and read permissions are set to Everyone. See Share jobs and export results.
- Change the settings on the computer to extend the time before the computer goes into sleep or hibernate mode.

Administering jobs

Users with the Admin role, or a role with equivalent capabilities, can restrict how many jobs a given user can run, and how much space their job artifacts can take up.

You must define a role with the desired restrictions and assign the user to the role. You can apply a high level of granularity by giving unique roles to each user in your system.

Edit search restriction settings

To edit the search restrictions setting for a role:

1. In Splunk Web, go to **Settings > Access Controls > Roles**. In Splunk Enterprise you can manually edit search restrictions, which are specified in the `authorize.conf` file, as described in **Edit search restrictions manually**.

You can manage running jobs from the command line. For more information, see Manage search jobs from the OS.

Edit search restrictions manually

1. Review the contents of the authorize.conf.example file in the *Admin Manual*. This example explains some the attributes that you might want to use.
2. Create the configuration file.

Scope	Description
System-wide	Create the `authorize.conf` file in local directory for the system. The location of the system local directory is `$SPLUNK_HOME/etc/system/local` .
Application-specific	Create the `authorize.conf` file in the local directory for the application. The location of an application local directory is `$SPLUNK_HOME/etc/apps/<app_name>/local`.

3. Edit the local `authorize.conf` file. To restrict the jobs that users can run, add the following information to the file:
    1. Add a stanza for the role that you want to create. Use the format `[role_<roleName>]`. Role names must be in lowercase characters. For example, `[role_ninja]`.
    2. Optional. Add the `importRoles` attribute. Importing a role also imports the other aspects of that role, such as the indexes that the role is allowed to search. For example, `importRoles = user`.
    3. Add the `srchDiskQuota` attribute and value. This is the maximum amount of disk space (MB) that search jobs can use, for a user that belongs to this role. The default value is 100MB. For example, `srchDiskQuota = 500`.
    4. Add the `srchJobsQuota` attribute and value. This is the maximum number of concurrently running searches that a user of this role can

have. The default value is 3. `srchJobsQuota = 10`.
5. Optional. Add the `rtsearch` attribute to specify if the user is authorized to run **real-time searches**. If you enable real-time searches for the user, you should also specify the `rtSrchJobsQuota` attribute.

- For attribute descriptions and information about the default values, see role_name stanza for the `authorize.conf` file in the *Admin Manual*.
- For more information about roles, see Add and edit roles in the *Securing Splunk Enterprise* manual.

See also

- Create and edit reports in the *Reporting Manual*

Extending job lifetimes

When you run a new search job, the job is retained in the system for a period of time, called the job **lifetime**. During the lifetime, you can access the job and view the data returned by the job. If the job is not accessed within the specified lifetime, the job expires and is removed from the system.

There are two lifetime settings, 10 minutes and 7 days. The lifetime starts from the moment the job is run.

Default job lifetimes

The default lifetime for a search job depends on whether the search job is an artifact of an unscheduled or **scheduled search**.

For example, dashboard panels that are based on an inline search, use unscheduled searches. Panels that are based on a report, use saved searches. The saved search can be unscheduled or scheduled.

Default lifetimes for unscheduled searches

When you run an **ad hoc search** and the search is finalized or completes on its own, the resulting search job has a default lifetime of 10 minutes. Other

knowledge objects, such as real-time alerts and panels based on inline searches that use unscheduled searches have the same default lifetime.

Default lifetimes for scheduled searches

Scheduled searches launch search jobs on a regular interval. By default, these jobs are retained for the interval of the scheduled search multiplied by two. For example, if the search runs every 6 hours, the resulting jobs expire in 12 hours.

Automatic lifetime extensions

Whenever you access an active job, such as when you view the results of a search job, the lifetime is reset. The reset happens whether the job lifespan is 10 minutes or 7 days. Here are a few examples of how this works.

- If the lifetime is set to 10 minutes and you run the search job at 11:00 AM, the job lifetime is set to end at 11:10 AM. If you run the job again at 11:07 AM, the job lifetime is reset to end at 11:17 AM.
- If you set the lifetime for a job to 7 days and then access the job 4 days later, the job lifetime is reset and will not expire for another 7 days from the current day and time.

Changing the lifetime for the current job

You change the lifetime setting for the current ad hoc search job in the Search app.

1. Select the **Job** drop-down.
2. Select **Edit Job Settings** to display the Job Settings.
3. For **Lifetime**, select either 10 Minutes or 7 Days.

Changing the lifetime for active jobs

You can change the lifetimes for jobs resulting from previously run unscheduled or scheduled searches. You can only change the lifetimes for active search jobs. See Manage search jobs.

Expired jobs

After the job lifetime ends, the job expires and is deleted from the system.

It is possible that while you are looking at the list of jobs that a job will expire. When you try to extend the lifetime of the expired job, a message appears explaining that the job no longer exists. You cannot extend the lifetime of an expired job.

Changing the default lifetime values

You can change the default value for the job lifetime for unscheduled and scheduled searches.

Change the default lifetime value for unscheduled searches

In Splunk Enterprise, you can change the default value for the job lifetime for unscheduled searches.

Prerequisites

- Only users with file system access, such as system administrators, can change the default lifetime values.
- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

1. Open the local `limits.conf` file for the Search app. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. In the `[search]` stanza, change the `default_save_ttl` value to a number that is appropriate for your needs. The acronym TTL is an abbreviation for "time to live."

If you are using Splunk Cloud and want to change the default job lifetime value for unscheduled searches, open a Support ticket.

Change the default lifetime value for scheduled searches

In Splunk Enterprise, you can change the default lifetime for jobs resulting from a specific scheduled search.

Prerequisites

- Only users with file system access, such as system administrators, can change the default lifetime values.

- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

1. Open the local `savedsearches.conf` file. For example, `$SPLUNK_HOME/etc/apps/<app_name>/local`.
2. Locate the scheduled search, and change the `dispatch.ttl` setting to a different interval multiple.

If you are using Splunk Cloud and want to change the default job lifetime value for scheduled searches, open a Support ticket.

Share jobs and export results

You can share a job with other Splunk users, or export the event data to archive or to use with a third-party charting application.

Share a job with others

When you share a job, you are sharing the results of a specific run of a search.

There are several ways that you can share a specific job with other Splunk users. You can change the permissions for a search job to share that job with other users. You can also share a job by sending the URL for a search job to a Splunk user.

You can only change permissions or share a link to the current job.

Change job permissions

You can share a job by changing the permissions on that job. By default, all jobs are **Private**.

1. From the **Job** menu, select **Edit Job Settings** to display the Job Settings dialog box.
2. Change **Read Permissions** to **Everyone**.
3. Click **Save**

Share a job URL

You can share a job with other Splunk users by sending them a link to the job. This is handy when you want another user to see the results returned by the job.

The users that you send the link to must have permissions to use the app that the job belongs to.

Decide which method you want to use to obtain a job link. You can use the **Share** icon or the **Job** menu.

1. To use the Share icon:
    1. Click the icon. The Share icon is one of the search action icons.
    2. In the **Link To Job** text box, copy the URL and send the link to the users that you want to share the job results with.

       The permissions on the job are automatically changed to **Everyone** and the lifetime of the job is automatically extended to **7 days**.

- To use the **Job** menu:

    1. From the **Job** menu, select **Edit Job Settings** to display the Job Settings dialog box.

2. Change **Read Permissions** to **Everyone**. If the permissions for a job are set to **Private**, other users cannot access the job with the link.
3. Change **Lifetime** to **7 days**.
4. Copy the link and send the link to the users you want to share the job results with.

You can also save the link for your own use by using the Bookmark icon. The Bookmark icon appears in both the Job Settings dialog box and the Share Jobs dialog box. You can click and drag the Bookmark icon to the bookmarks bar in your Web browser.

Export job results to a file

You can export your job results in a variety of format such as CSV, JSON, PDF, Raw Events, and XML. You can then archive the file, or use the file with a third-party charting application. The format options depend on the type of job artifact that you are working with.

- If the search generates calculated data that appears on the Statistics tab, you cannot export using the Raw Events format.
- If the search is a saved search, such as a Report, you can export using the PDF format.

The export file is saved in the default download directory for your browser or operating system.

There are several methods that you can use to export search results. A few of these methods include Splunk Web, CLI, SDKs, and REST. Some of the methods are optimized for speed, while others are good for extremely large event sets.

For a complete list of the export methods and links to the specific steps, see Export search results.

Manage search jobs

You can use the Jobs page to review and manage any **job** that you own.

If you have the Admin role, or a role with an equivalent set of capabilities, you can manage the search jobs run by all users of your Splunk implementation.

Opening the Jobs page

- In Splunk Web, to view a list of your jobs select **Activity > Jobs**. This opens the **Jobs** page.

The Jobs page displays a list of different types of search jobs.

- Jobs resulting from ad hoc searches or pivots that you have recently run manually.
- Jobs for searches that are run when dashboards are loaded or reports are opened.
- Jobs for scheduled searches.

Refreshing the jobs list

The list of jobs in the Jobs page does not automatically refresh.

- Jobs that are created after you display the Jobs page are not visible until

you reload the Jobs page.
- If a job expires while you have the Jobs page open, the job appears in the Jobs page list, but you cannot view the job results.

To refresh the Jobs page, reload the page.

Job actions

You can use the Actions column to perform actions on a job.

Use the **Job** drop-down to edit the job settings, extend the job lifetime, inspect the job, or delete the job.

Use the action icons to pause, stop, share, and export jobs.

To perform these actions on multiple jobs, select the jobs and click **Edit Selected**. Then select the action that you want to perform.

View and compare jobs

You can see a list of the jobs you have recently dispatched or saved for later review. Use the list to compare job statistics such as run time, total count of events matched, size, and so on.

Active job count

In the upper right corner of the Jobs page there is a count of the total number of jobs in the list.

The count reflects the number of jobs at the moment you opened the Jobs page. If a job expires while the Jobs page is open, the job count does not refresh.

Sort the job list

By default, the list of jobs is sorted by the **Created at** column.

You can sort the list by any column that displays a sort button in the column heading. For example, you can sort the list by the job expiration or by the job

owner.

- Click once on the column heading to sort the list in ascending order. Click again to sort the list in descending order.

Filter the job list

You can filter the list of jobs by application, by owner, and by status.

- In the **Filter** box, type a term or expression that appears in the search criteria to filter the list.

For example, you can specify `diskUsage`, `EMBED AND diskUsage=8*`, or `label=EMBED AND diskUsage=8*` in the Filter box.

View job search results

You can view the results of a search that is listed on the Jobs page.

1. Click on the search link to view the results associated with a specific job.
   - For ad hoc searches the link is the search criteria.
   - For saved searches the link is the name of the report, dashboard panel, or pivot.

   The results open in the Search app view.

Check the progress of ongoing jobs

You can check on jobs that are dispatched by scheduled searches, real-time searches, and long-running historical searches.

Use the **Status** column to check on the progress of ongoing jobs. The **Status** column shows the percent of the events that have been processed. Current jobs have a status of **Running**. Jobs that are running in the background have a status of **Backgrounded**.

Change the per page job count

You can change the number of jobs that appear on each page in the list. The default is to display ten jobs on each page. On the right side of the window, you can display 10, 20 or 50 jobs per page.

Inspect jobs

You can inspect a job to take a closer look at what a search is doing and see where the Splunk software is spending most of its processing time.

Use the Search Job Inspector to view information about the current job, such as job execution costs and search job properties.

1. In the **Actions** column for the specific job, select **Job**.
2. Select **Inspect Job**.

For more information about the using the Search Job Inspector, see View search job properties.

Extending search job lifetimes

From the Jobs page, there are several ways to change the lifetime of a job. To learn more about lifetimes for specific types of jobs, see Extending job lifetimes.

Quickly extend job lifetimes

You can quickly extend the lifetime of a job.

1. In the **Actions** column for the specific job, select **Job**.
2. Select **Extend Job Expiration**.

Extend the lifetimes for multiple jobs

You can extend the lifetimes of multiple jobs at the same time.

1. Select the jobs whose lifetimes you want to extend.
2. Above the job list, click **Edit Selected**.
3. Select **Extend Expiration**.

Choose the period of time for the lifetime extension

Use this method to select the time period for the extension.

1. In the **Actions** column for the specific job, select **Job**.
2. Select **Edit Job Settings**.
3. Select either 10 minutes or 7 days.
4. Click **Save**.

Share jobs

You can share a job by changing the job permissions, or by sharing a URL to the job.

Change job permissions

You can share a job by changing the permissions on that job. By default, all jobs are set to Private.

1. From the **Job** menu, select **Edit Job Settings** to display the Job Settings dialog box.
2. Change **Read Permissions** to **Everyone**.
3. Click **Save**.

Share a job URL

You can share a job with other Splunk users by sending them a link to the job.

The users that you send the link to must also have permissions to use the app that the job belongs to.

1. In the **Actions** column, click the Share icon for the job that you want to share.
2. In Share Job dialog box, copy the URL in the **Link To Job**.
3. Send the link to the users that you want to share the job results with.

> The permissions on the job are changed to **Everyone** and the lifetime of the job is extended to **7 days**.

You can also obtain the URL from the **Job Settings** window. If you use this method to share a job, you must change the **Read Permissions** to **Everyone** and the **Lifetime** to **7 days**.

Export jobs

You can export the event data from a job in a variety of formats such as CSV, JSON, PDF, Raw Events, and XML. You can then archive the file, or use the file with a third-party charting application. The format options depend on the type of job artifact that you are working with.

1. In the **Actions** column, click the Export icon for the event data that you want to export.
2. Click **Format** and select the format that you want the search results to be exported in.
3. In the **File Name** field, type a name for the export file where the event data will be stored.
4. In the **Number of Results** field, specify the number of results that you want to export.
5. Click **Export** to save the job events in the export file.

The export file is saved in the default download directory for your browser or operating system.

Note: If your search returns a large number of results, you can access all of the results in the Search app. However, the full set of results might not be stored with the **search job artifact**. When you export search results, the export process is based on the search job artifact, not the results in the Search app. If the artifact does not contain the full set of results, a message appears at the bottom of the Export Results dialog box to tell you that the search will be rerun by the Splunk software before the results are exported.

For more information, see Export search results.

If the Export icon is not visible, the icon has been hidden by your system administrator to prevent data export.

Extend the session timeout when exporting large amounts of data

When you try to export large amounts of data using the Export button, the session might timeout.

Users with the Admin role, or a role with an equivalent set of capabilities, can use the following procedure to extend the session timeout limit.

1. Click **Settings > Server Settings > General Settings**.

2. In the **Splunk Web** section, increase the number in the Session timeout field.
3. Click **Save**.

Increasing the timeout settings allows Splunk Web more time for the connection between your browser and Splunk Web.

Delete jobs

You can delete one or more jobs from the jobs list.

Delete a single job

You can delete a job from the job list.

1. In the **Actions** column for the specific job, select **Job**.
2. Select **Delete Job**.

Delete multiple jobs

You can delete multiple jobs at one time.

1. Select the jobs that you want to delete.
2. Above the job list, click **Edit Selected**.
3. Select **Delete**.

View search job properties

The **Search Job Inspector** is a tool that lets you take a closer look at what your search is doing and see where the Splunk software is spending most of its time.

This topic discusses how to use the Search Job Inspector to both troubleshoot the performance of a search **job** and understand the behavior of knowledge objects such as event types, tags, lookups and so on within the search. For more information, see Manage search jobs in this manual.

Here's a Splunk Education video about Using the Splunk Search Job Inspector. This video shows you how to determine if your search is running efficiently, event types, searches in a distributed environment, search optimization, and disk usage.

Open the Search job inspector

You can access the Search job inspector for a search job, as long as the search job has not expired (which means that the search artifact still exists). The search does not need to be running to access the Search job inspector.

1. Run the search.

2. From the **Job** menu, select **Inspect Job**.

This opens the Search job inspector in a separate window.

View the properties of a search job

You can use the URL to inspect a search job artifact if you have its search ID (SID). You can find the SID of a search in the Job Manager (click the **Jobs** link in the upper right hand corner) or listed in Splunk's dispatch directory, `$SPLUNK_HOME/var/run/splunk/dispatch` . For more information about the Job Manager, see Manage search jobs in this manual.

If you look at the URI path for the Search Job Inspector window, you will see something like this at the end of the string:

```
.../manager/search/job_inspector?sid=1299600721.22
```

The `sid` is the SID number. The `namespace` is the name of the app that the SID is associated with. In this example, the SID is 1299600721.22.

Type the search artifact SID in the URI path, after `sid=` and press **Enter**. As long as you have the necessary ownership permissions to view the search, you will be able to inspect it.

Now, what exactly are you looking at?

What the Search Job Inspector shows you

At the top of the Search Job Inspector window, an information message appears. The message depends on whether the job is paused, running, or finished. For example, if the job is finished the message tells you how many results it found and the time it took to complete the search. Any error messages are also displayed at the top of the window.

The key information that the Search Job Inspector displays are the execution costs and the search job properties.

Execution costs
: The Execution costs section lists information about the components of the search and how much impact each component has on the overall performance of the search.

Search job properties
: The Search job properties section lists other characteristics of the job.

Execution costs

With the information in the **Execution costs** section, you can troubleshoot the efficiency of your search. You can narrow down which processing components are impacting the search performance. This section contains information about the search processing components that were used to process your search.

- The component durations in seconds.
- How many times each component was invoked while the search ran.
- The input and output event counts for each component.

The Search Job Inspector lists the components alphabetically. The number of components that you see depend on the search that you run.

The following tables describes the significance of each individual command and distributed component in a typical keyword search.

Execution costs of search commands

In general, for each command that is part of the search job, there is a parameter `command.<command_name>`. The values for these parameters represent the time spent in processing each `<command_name>`. For example, if the table command is used, you will see `command.table`.

Search command component name	Description
`command.search`	After the Splunk software identifies the events that contain the indexed fields matching your search, the events are analyzed to identify which events match the other search criteria. These are concurrent operations,

	not consecutive. • `command.search.index` - tells how long it took to look into the TSIDX files for the location to read in the raw data. This is the time spent identifying, from the tokens in the base search, what events to retrieve. • `command.search.rawdata` - tells how long it took to read the actual events from the rawdata files. • `command.search.typer` - tells how long it took to assign event types to events. • `command.search.kv` - tells how long it took to apply field extractions to the events. • `command.search.fieldalias` - tells how long it took to rename fields based according to `props.conf`. • `command.search.lookups` - tells how long it took to create new fields based on existing fields (perform field lookups). • `command.search.filter` - tells how long it took to filter out events that do not match, for example fields and phrases. • `command.search.tags` - tells how long it took to assign tags to events.

There is a relationship between the type of commands used and the numbers you can expect to see for Invocations, Input count, and Output count. For searches that generate events, you expect the input count to be 0 and the output count to be some number of events X. If the search is both a generating search and a filtering search, the filtering search would have an input (equal to the output of the generating search, X) and an output=X. The total counts would then be input=X, output=2*X, and the invocation count is doubled.

Execution costs of dispatched searches

Distributed search component name	Description
`dispatch.check_disk_usage`	The time spent checking the disk usage of this job.
`dispatch.createdSearchResultInfrastructure`	The time to create and set up the collectors for each peer and execute the HTTP post to each peer.
`dispatch.earliest_time`	

	Specifies the earliest time for this search. Can be a relative or absolute time. The default is an empty string.
dispatch.emit_prereport_files	When running a **transforming search**, Splunk Enterprise cannot compute the statistical results of the report until the search completes. After it fetches events from the search peers (dispatch.fetch), it, writes out the results to local files. dispatch.emit_prereport_files provides the time that it takes for Splunk Enterprise to write the transforming search results to those local files.
dispatch.evaluate	The time spent parsing the search and setting up the data structures needed to run the search. This component also includes the time it takes to evaluate and run subsearches. This is broken down further for each search command that is used. In general, dispatch.evaluate.<command_name> tells you the time spent parsing and evaluating the <command_name> argument. For example, dispatch.evaluate.search indicates the time spent evaluating and parsing the searchcommand argument.
dispatch.fetch	The time spent by the search head waiting for or fetching events from search peers. The dispatch.fetch value is different than the command.search value. The command.search value includes time spent by all indexers, which can be greater than the actual elapsed time of the search. If you have only a single node, then the dispatch.fetch and the command.search values will be similar. In a distributed environment, depending on the search, these values can be very different.
dispatch.preview	The time spent generating preview results.
dispatch.process_remote_timeline	The time spent decoding timeline information generated by search peers.
dispatch.reduce	

	The time spend reducing the intermediate report output.
`dispatch.stream.local`	The time spent by search head on the streaming part of the search.
`dispatch.stream.remote`	The time spent executing the remote search in a distributed search environment, aggregated across all peers. Additionally, the time spent executing the remote search on each remote search peer is indicated with: `dispatch.stream.remote.<search_peer_name>.output_count` represents bytes sent rather than events in this case.
`dispatch.timeline`	The time spent generating the timeline and fields sidebar information.
`dispatch.writeStatus`	The time spent periodically updating `status.csv` and `info.csv` in the job's dispatch directory.
`startup.handoff`	The time elapsed between the forking of a separate search process and the beginning of useful work of the forked search processes. In other words it is the approximate time it takes to build the search apparatus. This is cumulative across all involved peers. If this takes a long time, it could be indicative of I/O issues with .conf files or the dispatch directory.

Search job properties

The **Search job properties** fields provide information about the search job. The **Search job properties** fields are listed in alphabetical order.

Parameter name	Description
`cursorTime`	The earliest time from which no events are later scanned. Can be used to indicate progress. See description for `doneProgress`.
`delegate`	For saved searches, specifies jobs that were started by the user. Defaults to scheduler.
`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, or DONE.
`doneProgress`	A number between 0 and 1.0 that indicates the approximate progress of the search. `doneProgress = (latestTime ? cursorTime) / (latestTime ? earliestTime)`
`dropCount`	For real-time searches only, the number of possible events that were dropped due to the `rt_queue_size` (defaults to 100000).
`earliestTime`	The earliest time a search job is configured to start. Can be used to indicate progress. See description for `doneProgress`.
`eai:acl`	Describes the app and user-level permissions. For example, is the app shared globally, and what users can run or view the search?
`eventAvailableCount`	The number of events that are available for export.
`eventCount`	The number of events returned by the search. In other words, this is the subset of scanned events (represented by the `scanCount`) that actually matches the search terms.
`eventFieldCount`	The number of fields found in the search results.
`eventIsStreaming`	Indicates if the events of this search are being streamed.
`eventIsTruncated`	Indicates if events of the search have not been stored, and thus not available from the events endpoint for the search.
`eventSearch`	Subset of the entire search that is before any transforming commands. The timeline and events endpoint represents the result of this part of the search.
`eventSorting`	Indicates if the events of this search are sorted, and in which order. `asc` = ascending; `desc` = descending; `none` = not sorted
`isBatchMode`	Indicates whether or not the search in running in batch mode. This applies only to searches that include transforming commands.
`isDone`	Indicates if the search has completed.

`isFailed`	Indicates if there was a fatal error executing the search. For example, if the search string had invalid syntax.
`isFinalized`	Indicates if the search was finalized (stopped before completion).
`isPaused`	Indicates if the search has been paused.
`isPreviewEnabled`	Indicates if previews are enabled.
`isRealTimeSearch`	Indicates if the search is a real time search.
`isRemoteTimeline`	Indicates if the remote timeline feature is enabled.
`isSaved`	Indicates that the search job is saved, storing search artifacts on disk for 7 days from the last time that the job has been viewed or touched. Add or edit the `default_save_ttl` value in `limits.conf` to override the default value of 7 days.
`isSavedSearch`	Indicates if this is a saved search run using the scheduler.
`isTimeCursored`	Specifies if the cursorTime can be trusted or not. Typically this parameter it set to true if the first command is search.
`isZombie`	Indicates if 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`	Custom name created for this search.
`latestTime`	The latest time a search job is configured to start. Can be used to indicate progress. See description for `doneProgress`.
`numPreviews`	Number of previews that have been generated so far for this search job.
`messages`	Errors and debug messages.
`optimizedSearch`	The restructured syntax for the search that was run. The built-in optimizers analyze your search and restructure the search syntax, where possible, to improve search performance. The search that you ran is displayed under the **search** job property.
`performance`	This is another representation of the **Execution costs**.
`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.
`resultIsStreaming`	Indicates if 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`	Time in seconds that the search took to complete.
`scanCount`	The number of events that are scanned or read off disk.
`search`	The search string.
`searchCanBeEventType`	If the search can be saved as an event type, this will be 1, otherwise, 0. Only base searches (no subsearches or pipes) can be saved as event types.
`searchProviders`	A list of all the search peers that were contacted.
`sid`	The search ID number.
`statusBuckets`	Maximum number of timeline buckets.
`ttl`	The time to live, or time before the search job expires after it completes.
Additional info	Links to further information about your search. These links may not always be available. • timeline • field summary • search.log

Note: When troubleshooting search performance, it's important to understand the difference between the `scanCount` and `resultCount` costs. For dense searches, the `scanCount` and `resultCount` are similar (`scanCount = resultCount`); and for sparse searches, the `scanCount` is much greater than the result count (`scanCount >> resultCount`). Search performance should not so much be measured using the `resultCount`/time rate but `scanCount`/time instead. Typically, the `scanCount`/second event rate should hover between 10k and 20k events per second for performance to be deemed good.

Debug messages

Configure the Search Job Inspector to display DEBUG messages when there are errors in your search. For example, DEBUG messages can warn you when there are fields missing from your results.

The Search Job Inspector displays DEBUG messages at the top of the Search Job Inspector window, after the search has completed.

By default the Search Job Inspector hides DEBUG messages. To configure their display, open `limits.conf` and set the `infocsv_log_level` parameter in the `[search_info]` stanza to `INFO`.

```
[search_info]
infocsv_log_level = INFO
```

Examples of Search Job Inspector output

Here's an example of the execution costs for a dedup search, run over All time:

```
* | dedup punct
```

The search commands component of the **Execution costs** panel might look something like this:

The command.search component, and everything under it, gives you the performance impact of the `search` command portion of your search, which is everything before the pipe character.

The `command.prededup` gives you the performance impact of processing the results of the `search` command before passing it into the `dedup` command.

- The Input count of `command.prededup` matches the Output count of `command.search`.
- The Input count of `command.dedup` matches the Output count of `command.prededup`.

In this case, the Output count of `command.prededup` should match the number of events returned at the completion of the search. This is the value of `resultCount`, under **Search job properties**.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the Search Job Inspector.

Dispatch directory and search artifacts

Each search or alert you run creates a **search artifact** that must be saved to disk. The artifacts are stored in directories under the `dispatch` directory. For each **search job**, there is one search-specific directory. When the job expires, the search-specific directory is deleted.

See About jobs and job management for information about search jobs.

Dispatch directory location

The `dispatch` directory stores artifacts on nodes where the searches are run. The nodes include search heads, search peers, and standalone Splunk Enterprise instances. The path to the `dispatch` directory is `$SPLUNK_HOME/var/run/splunk/dispatch`.

Dispatch directory contents

In the `dispatch` directory, a search-specific directory is created for each search or alert. Each search-specific directory contains a CSV file of its search results, a `search.log` file with details about the search execution, and more. These are 0-byte files.

View dispatch directory contents

From a command-line window, or UI window such as Windows Explorer or Finder, you can list the search-specific directories.

For example to view a list in a command-line window, change to the dispatch directory and list the contents in that directory. The following list contains both ad hoc, real-time, and scheduled search-specific directories.

```
# cd $SPLUNK_HOME/var/run/splunk/dispatch
# ls
1346978195.13
rt_scheduler__admin__search__RMD51cfb077d0798f99a_at_1469464020_37.0
1469483311.272
1469483309.269
1469483310.27
scheduler__nobody_c3BsdW5rX2FyY2hpdmVy__RMD5473cbac83d6c9db7_at_1469503020_53
admin__admin__search__count_1347454406.2
rt_1347456938.31
1347457148.46
subsearch_1347457148.46_1347457148.1
```

The contents of a search-specific directory includes files and subdirectories. The following example shows the contents of a search-specific directory named `1346978195.13`.

```
#ls 1346978195.13/
args.txt
audited
buckets/
events/
generate_preview
info.csv
metadata.csv
peers.csv
request.csv
results.csv.gz
runtime.csv
search.log
status.csv
timeline.csv
```

Windows users should use **dir** instead of **ls** in a command-line window, or Windows Explorer, to see the contents of the `dispatch` directory.

File descriptions for search-specific directories

The files or subdirectories that appear in a search-specific directory depend on the type of search that you run. The following table lists the files and subdirectories that might appear in your search-specific directories.

File name	Contents
args.txt	The arguments that are passed to the search process.
alive.token	The status of the search process. Specifies if the search is alive or not alive.
audited	A flag to indicate the events have been audit signed.
buckets	A subdirectory that contains the the field picker statistics for each-bucket. The buckets are typically the chunks that are visible in the search histogram UI. This is not related to index buckets.
custom_prop.csv	Contains custom job properties, which are arbitrary key values that can be added to search job, retrieved later, are mostly used by UI display goals.
events	A subdirectory that contains the events that were used to generate the search results.
generate_preview	A flag to indicate that this search has requested preview. This file is used mainly for Splunk Web searches.
info.csv	A list of search details that includes the earliest and latest time, and the results count.
metadata.csv	The search owner and roles.
peers.csv	The list of peers asked to run the search.
pipeline_sets	The number of pipeline sets an indexer runs. The default is 1.
remote_events or events_num_num.csv.gz	Used for the remote-timeline optimization so that a reporting search that is run with `status_buckets>0` can be MapReduced.
request.csv	A list of search parameters from the request, including the fields and the text of the search.
results.csv.gz	An archive file that contains the search results.
rtwindow.czv.gz	Events for the latest real-time window when there are more events in the window than can fit in memory. The default limit is 50K.
runtime.csv	Pause and cancel settings.

search.log	The log from the search process.
sort?	A sort temporary file, used by the `sort` command for large searches.
srtmpfile?	A generic search tempfile, used by facilities which did not give a name for their temporary files.
status.csv	The current status of the search, for example if the search is still running. Search status can be any of the following: QUEUED, PARSING, RUNNING, PAUSED, FINALIZING, FAILED, DONE.
timeline.csv	Event count for each timeline bucket.

The dispatch directory also contains ad hoc data model acceleration summaries. These are different from persistent data model acceleration summaries, which are stored at the index level.

Dispatch directory naming conventions

The names of the search-specific directories in the dispatch directory are based on the type of search. For saved and scheduled searches, the name of a search-specific directory is determined by the following conditions.

- The name of the search is less than 20 characters and contains only ASCII alphanumeric characters. The search-specific directory name includes the search name.
- The name of the search is 20 characters or longer, or contains non-alphanumeric characters. A hash is used for the name. This is to ensure a search-specific directory named by the search ID can be created on the filesystem.

A search that contains multiple subsearches might exceed the maximum length for the dispatch directory name. When the maximum length is exceeded, the search fails.

Type of search	Naming convention	Examples
Local ad hoc search	The UNIX time of the search.	An ad hoc search. `1347457078.35` A real-time ad hoc search. `rt_1347456938.31` An ad hoc search that uses a subsearch, which creates two dispa `1347457148.46 subsearch_1347457148.46_1347457148.1`

Saved search	The user requesting the search, the user context the search is run as, the app the search came from, the search string, and the UNIX time.	?count? ? run by **admin**, in user context **admin**, saved in app **se**
		
		`admin__admin__search__count_1347454406.2`
		
		?Errors in the last 24 hours? ? run by **somebody**, in user context saved in app **search**
		
		`somebody__somebody__search_RMD5473cbac83d6c9db7_134745513`

Scheduled search	The user requesting the search, the user context the search is run as, the app the search came from, the search string, UNIX time, and an internal ID added at the end to avoid name collisions.	?foo? ? run by the **scheduler**, with no user context, saved in app
		`scheduler__nobody__unix__foo_at_1347457380_051d958b8354c5`
		
		?foo2? - remote peer search on search head **sh01**, with **admin** u by the **scheduler**, saved in app **search**
		
		`remote_sh01_scheduler__admin__search__foo2_at_1347457920_`

Remote search	Searches that are from remote peers start with the word "remote".	?foo2? - remote peer search on search head **sh01** `remote_sh01_scheduler__admin__search__foo2_at_1347457920_`
Real-time search	Searches that are in real-time start with the letters "rt".	Ad hoc real-time search `rt_1347456938.31`
Replicated search	Searches that are replicated search results (artifacts) start with "rsa_".These SIDs occur in search head clusters when a completed search	

	is replicated to a search head other than the search head that originally ran the search	
Replicated scheduled search	The "rsa_scheduler_" prefix is for replicated results of searches dispatched by the scheduler.	The search ?foo? is run by the **rsa_scheduler**, with no user cont **unix**. `rsa_scheduler_nobody_unix_foo_at_1502989576_051d958b`
Report acceleration search	These are the probe searches created by datamodel acceleration to retrieve the acceleration percentage from all peers.	`SummaryDirector_1503528948.24878_D12411CE-A361-4F75-B90A-`

Dispatch directory maintenance

The dispatch directory reaper iterates over all of the artifacts every 30 seconds. The reaper deletes artifacts that have expired based, on the last time that the artifacts were accessed and their configured time to live (TTL), or lifetime.

See Extending job lifetimes for information about changing the default lifetime for the search artifact using Splunk Web.

Search artifact lifetime in the dispatch directory

Default lifetime values depend on the type of search. The lifetime countdown begins when the search completes. The following table lists the default lifetime values by type of search.

Search type	Default lifetime
Manually run saved search or ad hoc	10 minutes

search	
Remote search from a peer	10 minutes
Scheduled search	The lifetime varies by the selected alert action, if any. If an alert has multiple actions, the action with the longest lifetime becomes the lifetime for the search artifact. Without an action, the value is determined by the `dispatch.ttl` attribute in the `savedsearches.conf` file,. The default value for the `dispatch.ttl` attribute is twice (2x) the scheduled period. Alert actions determine the default lifetime of a scheduled search. • Email, RSS, and tracking alert actions have a default lifetime of 24 hours. • Script alert actions have a default lifetime of 10 minutes. • Summary indexing and populate lookup alert actions have a default lifetime of 2 minutes.
Show source scheduled search	30 seconds
Subsearch	5 minutes Subsearches generate two search-specific directories. There is a search-specific directory for the subsearch and a search-specific directory for the search that uses the subsearch. These directories have different lifetime values.

Change search artifact lifetime

There are a number of ways to change the search artifact lifetime. Modifying the default search behavior affects searches with no other lifetime value or TTL applied.

See How to edit a configuration file.

Search behavior type	Process
Global search behavior	In the `limits.conf` file, set `ttl` or `remote_ttl` in the [search] stanza or `ttl` in the [subsearch] stanza.

	Modifying `limits.conf` provides the default for searches, so it affects searches with no other lifetime value applied.
Search specific behavior	In the `savedsearches.conf` file, set `dispatch.ttl` for an individual search. Or set an individual value for a search when you save the search in Splunk Web. This overrides the default search behavior.
Searches with alert actions	In the `alert_actions.conf`, set a `ttl` value to specify the minimum lifetime of a search artifact if a certain alert action is triggered. This overrides any shorter lifetime applied to a search.

Clean up the dispatch directory based on the age of directories

As more and more artifacts are added to the dispatch directory, it is possible that the volume of artifacts will cause a adverse effect on search performance or that a warning appears in the UI. The warning threshold is based on the the `dispatch_dir_warning_size` attribute in the `limits.conf` file.

The default value for the `dispatch_dir_warning_size` attributes is 5000 jobs.

You can move search-specific directories from the dispatch directory to another, destination, directory. You move search-specific directories by using the `clean-dispatch` command. You must specify a time that is later than the last modification time for the search-specific directories. The destination directory must be on the same file system as the dispatch directory.

Run the command `$SPLUNK_HOME/bin/splunk clean-dispatch help` to learn how to use the `clean-dispatch` command.

See Too many search jobs in the *Troubleshooting Manual* for more information about cleaning up the dispatch directory.

Limit search process memory usage

Splunk software can be configured to automatically terminate search job processes that exceed a threshold of a configured quantity of resident memory in

use.

You might be interested in using this feature if:

- You want to be proactive and avoid a scenario where one runaway search causes one or several of your search peers to crash.
- You already have encountered this scenario and do not want it to happen again.
- In the Distributed Management Console, the **Search Activity: Instance** view exposes one or more searches that consume dangerous amounts of physical memory. You can see this information in the **Top 10 memory-consuming search** panel.

If you have Splunk Cloud and want to adjust this threshold, you must file a Support ticket, because you do not have access to the `limits.conf` file.

What does this threshold do?

Enabling this threshold limits the maximum memory permitted for each search process. A search process that is an outlier in memory size is automatically killed off, limiting damage.

This threshold uses process resource usage information that is recorded by platform instrumentation. So this feature works only on *nix, Solaris, and Windows platforms.

- See Introspection endpoint descriptions in the *REST API Reference Manual*.
- See About the platform instrumentation framework in the *Troubleshooting Manual*.

Search memory is checked periodically, so a rapid spike might exceed the configured limit.

The functionality is wired into the DispatchDirectoryReaper, so stalls in the reaper components also cause stalls in how often the memory of searches are checked.

Enable a search process memory threshold

The search process memory tracking is disabled by default.

1. See How to edit a configuration file in the *Admin Manual*.

2. Open the `limits.conf` file.

3. In the [search] stanza, change the setting for the `enable_memory_tracker` attribute to `true`.

4. Review and adjust the memory limit.

> You can set the limit to an absolute amount or a percentage of the identified system maximum, using `search_process_memory_usage_threshold` or `search_process_memory_usage_percentage_threshold`, respectively. Searches are always tested against both values, and the lower value applies. See limits.conf.spec in the **Admin Manual**.

5. To enable the configuration changes, restart Splunk Enterprise.

Where is threshold activity logged?

If the threshold causes a search process to be stopped on a search head, an error is inserted into the search artifact file `info.csv`. If the search is run through Splunk Web, this error message also appears in Splunk Web. The error states that the process was terminated and specifies the limit setting and value.

If the threshold causes a search process to be stopped on a search peer, a WARN message is logged in the `splunkd.log` file in the StreamedSearch category.

In both cases, a WARN message is logged in the `splunkd.log` file in the DispatchReaper category.

The messages are similar to:

```
Forcefully terminated search process with sid=... since
its \[relative physical or physical] memory usage (... \[MB or %])
has exceeded the \[relative physical or physical] memory
threshold specified in limits.conf/...setting name.. (...setting
value...)
```

Manage Splunk Enterprise jobs from the OS

If you have Splunk Enterprise on Microsoft Windows or *nix, you can manage search jobs from the operating system as described in this topic. For information on how to manage search jobs in Splunk Web, see Manage search jobs in this

manual.

Manage jobs in *nix

When a search job runs, it will manifest itself as a process in the OS called `splunkd search`. You can manage the job's underlying processes at the OS command line.

To see the job's processes and its arguments, type:

```
> top
> c
```

This will show you all the processes running and all their arguments.

Typing `ps -ef | grep "search"` will isolate all the Splunk search processes within this list. It looks like this:

```
[pie@fflanda ~]$ ps -ef | grep 'search'
530369338 71126 59262   0 11:19AM ??         0:01.65 [splunkd pid=59261]
search --id=rt_1344449989.64 --maxbuckets=300 --ttl=600 --maxout=10000
--maxtime=0 --lookups=1 --reduce_freq=10 --rf=* --user=admin --pro
--roles=admin:power:user  AhjH8o/Render TERM_PROGRAM_VERSION=303.2
530369338 71127 71126   0 11:19AM ??         0:00.00 [splunkd pid=59261]
search --id=rt_1344449989.64 --maxbuckets=300 --ttl=600 --maxout=10000
--maxtime=0 --lookups=1 --reduce_freq=10 --rf=* --user=admin --pro
--roles=
```

There will be two processes for each search job; the second one is a "helper" process used by the `splunkd` process to do further work as needed. The main job is the one using system resources. The helper process will die on its own if you kill the main process.

The process info includes:

- the search string (search=)
- the job ID for that job (id=)
- the ttl, or length of time that job's artifacts (the output it produces) will remain on disk and available (ttl=)
- the user who is running the job (user=)
- what role(s) that user belongs to (roles=)

When a job is running, its data is being written to `$SPLUNK_HOME/var/run/splunk/dispatch/<job_id>/` Scheduled jobs (scheduled saved searches) include the saved search name as part of the directory name.

The value of `ttl` for a process will determine how long the data remains in this spot, even after you kill a job. When you kill a job from the OS, you might want to look at its job ID before killing it if you want to also remove its artifacts.

Manage jobs in Windows

On Windows, each search likewise runs as a separate process. Windows does not have a command-line equivalent for the *nix `top` command, but there are several ways in which you can view the command line arguments of executing search jobs.

- Use the Process Explorer utility (http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx) to find the command line of the process that is performing the search.
- Use the `TASKLIST` and `Get-WMIObj` commands in the Powershell (http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx) command line environment to get the `ProcessID` and `CommandLine` arguments for a search job.

When a search runs, the data for that search is written into the `%SPLUNK_HOME\var\run\splunk\dispatch\<epoch_time_at_start_of_search>.<number_separator>` directory. Saved searches are written to similar directories that have a naming convention of "`admin__admin__search_`" and a randomly-generated hash of numbers in addition to the UNIX time.

Use the filesystem to manage jobs

You can manage a job by creating and deleting files in the job's artifact directory:

- To cancel a job, go into that job's artifact directory create a file called 'cancel'.
- To preserve that job's artifacts (and ignore its ttl setting), create a file called 'save'.
- To pause a job, create a file called 'pause', and to unpause it, delete the 'pause' file.

Save and Schedule Searches

Saving searches

When you create a search, you have several options to choose from to save the search. In the Search app, the choices are listed under the **Save As** drop-down.

Save as option	Description	More information
Report	When you create a search that you would like to run again, you can save the search as a report.	See Create and edit reports in the *Reporting Manual*.
Dashboard panel	You can also save a search as a dashboard panel. Dashboards can have one or more panels which can show search results in tables or in graphical visualizations.	See Getting started in the *Dashboards and Visualizations* manual.
Alert	Some searches provide timely information that you want to be notified about. You can save a search as an alert. An alert is an action that a saved search triggers, based on the results of the search. The action might be to send an email or run a script.	See About alerts in the *Alerting Manual*.
Event type	You can save a search as an event type. Event types are a categorization system to help you make sense of your data. Event types let you sift through huge amounts of data, find similar patterns, and create alerts and reports.	See About event types in the *Knowledge Manager Manual*.

See Also

Scheduling searches

Scheduling searches

You can **schedule searches** to run on a regular basis.

Option	Description	More information
Report	After you save a search as a report, you can convert that report into a scheduled report. A scheduled report is a report that runs on a scheduled interval, and which can trigger an action each time the report runs. There are two actions available for scheduled reports: Send email and Run a script.	See Schedule reports in the *Reporting Manual*.
Dashboard panel	There are several options to create a scheduled report: • You can create a dashboard panel that is based on that scheduled report. • When you save an **ad hoc search** as a dashboard panel, the panel refers to the search as an inline search. You can edit the dashboard panel to convert the search to a report and then schedule the report.	See Working with dashboard panels in the *Dashboards and Visualizations* manual.
Alert	You can create a scheduled alert to search for events on a regular schedule. You can configure scheduling, trigger conditions, and throttling to customize the alert.	See Create scheduled alerts in the *Alerting Manual*.

See Also

Saving searches

Export search results

Export search results

You can export search results from your Splunk deployment, and forward data to third-party systems, as described in this topic.

What are the available export methods?

The Splunk platform provides several export methods:

- Export data using Splunk Web
- Export data using the CLI
- Export data using SDKs
- Export data using REST API
- The `dump` search command
- Data forwarding

Splunk apps

- Deploy and Use Splunk App for CEF
- Deploy and Use Splunk DB Connect
- Hadoop Connect
- Install and Use Splunk ODBC Driver with Microsoft Excel
- Install and Use Splunk ODBC Driver with MicroStrategy
- Install and Use Splunk ODBC Driver with Tableau

Export options

The export method you choose depends on the data volumes involved and your level of interactivity. For example, a single on-demand search export through Splunk Web might be appropriate for a low-volume export. Alternatively, if you want to set up a higher-volume, scheduled export, the SDK and REST options work best.

For large exports, the most stable method of search data retrieval is the Command Line Interface (CLI). From the CLI, you can tailor your search to external applications using the various Splunk SDKs. The REST API works from the CLI as well, but is recommended only for internal use.

In terms of level of expertise, the Splunk Web and CLI methods are significantly more accessible than the SDKs and REST API, which require previous experience working with software development kits or REST API endpoints.

Method	Volume	Interactivity	Remarks
Splunk Web	Low	On-Demand, Interactive	Easy to obtain on-demand exports
CLI	Medium	On-Demand, Low Interactive	Easy to obtain on-demand exports
REST	High	Automated, best for computer-to-computer	Works underneath SDK
SDK	High	Automated, best for computer-to-computer	Best for automation

Supported export formats

You can export Splunk data into the following formats:

- Raw Events (for search results that are raw events and not calculated fields)
- CSV
- JSON
- XML
- PDF (for saved searches, using Splunk Web)

Export data using Splunk Web

You can export the event data from a search, report, or pivot job to various formats. You can then archive the file, or use the file with a third-party charting application.

1. After you run a search, report, or pivot, click the Export button. The Export button is one of the Search action buttons.

   If the button is not visible, it has been hidden by your system administrator to prevent data export.

Use the Export Results window to specify the format and name for your export file:

Sometimes your search must be run again before the results can be exported. See When exporting triggers your search to run again.

2. Click **Format** and select the format that you want the search results to be exported in.
   The supported formats depend on the type of job artifact that you are working with.

Format	Ad hoc searches	Saved searches	Notes
CSV	X	X	
JSON	X	X	
PDF		X	If the search is a saved search, such as a Report, you can export using the PDF format.
Raw Events	X	X	If the search generates calculated data that appears on the Statistics tab, you cannot export using the Raw Events format.
XML	X	X	

3. Optional. In the **File Name** field, you can type a name for the export file where the event data will be stored. If you do not specify a file name, a file is created using the search job ID as the file name. The search job ID is the UNIX time when the search was run. For example `1463687468_7.csv`.
4. Optional. In the **Number of Results** field, you can specify the number of results that you want to export. If you do not specify a number, all of the events are exported. For example, if you specify 500 in the **Number of Results** field, only the first 500 results returned from your search are exported.
5. Click **Export** to save the job events in the export file.

The file is saved in the default download directory for your browser or operating system. For example, for most Windows and Mac OS X users the export file

appears in the default **Downloads** directory. On Linux, check the XDG configuration file for the download directory.

When exporting triggers your search to run again

If your search returns a large number of results, it is possible that not all of the results will be stored with the **search job artifact**.

When you export search results, the export process is based on the search job artifact, not the results in the Search app. If the artifact does not contain the full set of results, a message appears at the bottom of the Export Results dialog box to tell you that the search will be rerun by the Splunk software before the results are exported.

The search is rerun when the search head believes that it cannot retrieve all of the events from the job artifact. The search head determines when to rerun the search based on the following logic:

- If the search is not a report, and one of the following is true.
  - The search is not done
  - The search is using a remote timeline
  - The search head believes that the search has not retained all of events

Extend the session timeout when exporting large amounts of data

When you export large amounts of data using the Export button, the session might timeout before the export is complete. You can extend the session timeout limit.

1. Click **Settings > Server Settings > General Settings**.
2. In the **Splunk Web** section, increase the number in the Session timeout field.
3. Click **Save**.

Increase the timeout setting to allow more time for the connection between your browser and Splunk Web.

Forward data to third-party systems

You can forward the data that you export to third-party systems.

- For an brief overview, see Forward data to third party systems in this manual.
- For more details, see Forward data to third party systems in *Forwarding Data*.

Use reports to send results to stakeholders

You can schedule reports to run on a regular interval and send the results to project stakeholders by email. The emails can present the report results in tables in the email, and as CSV or PDF attachments. The emails can also include links to the report results in Splunk Enterprise. See Schedule Reports in the *Reporting Manual*.

Export data using the CLI

The Command Line Interface (CLI) is easy to script, can handle automation, and can process volumes of data faster and more efficiently than Splunk Web. To access Splunk Enterprise through the CLI, you either need shell access to a Splunk Enterprise server, or permission to access the correct port on a remote Splunk server. If you have Splunk Cloud, you do not have shell access to your Splunk Cloud deployment, so cannot use the CLI to export data.

The syntax for exporting data using the CLI is as follows:

```
 splunk search [eventdata] -preview 0 -maxout 0 -output
[rawdata|json|csv|xml] > [myfilename.log] ...
```

By default, you can export a maximum of 100 events. To increase this number, use the `-maxout` argument. For example, if you include `-maxout 300000` you can export 300,000 events. Set `-maxout` to 0 to export an unlimited number of events.

To learn more about the Splunk Enterprise CLI, read About the CLI in the *Admin Manual*.

CLI output command example

This CLI example takes events from the `_internal` index that occur within the time range specified by the search string and outputs 200,000 of them in raw

data format to the file `test123.dmp`.

```
splunk search "index=_internal earliest=09/14/2015:23:59:00
latest=09/16/2015:01:00:00 " -output rawdata -maxout 200000 >
c:/test123.dmp
```

Export data using the Splunk REST API

Use the Splunk REST API to access data from the command line or a Web browser.

REST API access for Splunk Cloud deployments

If you have a self-service Splunk Cloud deployment and you want to use the Splunk REST API, file a Support ticket requesting the API to be enabled. See Using the REST API with Splunk Cloud in the *REST API Tutorials* for more details.

Export data

Exporting data starts with running a search job to generate results. You can then export this search result data to a file.

1. Run a search job using a POST to `/services/search/jobs/`. If you are using a custom time range, pass it in with the POST request.

   ```
   curl -k -u admin:changeme \
        https://localhost:8089/services/search/jobs/ -d
   search="search sourcetype=access_* earliest=-7d"
   ```
2. Get the search job ID (SID) for the search. The `/jobs` endpoint returns an XML response including the `<sid>`, or search job ID.

   ```
   <?xml version='1.0' encoding='UTF-8'?>
   <response>
     <sid>1423855196.339</sid>
   </response>
   ```
   You can also get the search job ID by viewing the job in the **Search Job Inspector**. Navigate to **Activity > Jobs** to open the Job Manager, locate the search job that you just ran, and click **Inspect**. The Search Job Inspector opens in a separate window.
3. Use a GET request on the `/results` endpoint to export the search results to a file. Ensure that you do the following in the GET request:
   - **Identify your object endpoints.**

To see a list of currently available object endpoints for your user, within your app, navigate to `https://localhost:8089/servicesNS/<user>/<app>/`. For example: `https://localhost:8089/servicesNS/admin/search/saved/searches/`

- **Identify the search job user and app.**
  The following example defines `<user>` as `admin` and `<app>` as `search`.
- **Identify an output format.**
  Use the `output_mode` parameter to specify one of the following available output formats. Use lower case for the format name, as shown here.
  `atom | csv | json | json_cols | json_rows | raw | xml`

This example exports search results to a JSON file.

```
curl -u admin:changeme \
     -k
https://localhost:8089/servicesNS/admin/search/jobs/1423855196.339/results/
\
     --get -d output_mode=json -d count=5
```

See also

For more details about the `/jobs` and `/export` endpoints, see the following information in the *REST API Reference*.

- search/jobs
- search/jobs/export

See also Creating searches using the REST API in the *REST API Tutorials*.

Export data using the Splunk SDKs

Splunk Software Development Kits (SDKs) enable software developers to create Splunk apps using common programming languages. Splunk SDKs let you integrate Splunk deployments with third-party reporting tools and portals, include search results in your application, and extract high volumes of data for archival purposes. To use Splunk SDKs, you should be proficient in SDK knowledge and development.

Splunk offers SDKs for Python, Java, JavaScript, and C#. When you run an export-search in these SDKs, the search runs immediately, it does not create a job for the search, and it start streaming results immediately.

The Splunk SDKs are built on top of the Splunk REST API. They provide a simpler interface for the REST API endpoints. With fewer lines of code, you can write applications that can:

- Create and run authenticated searches
- Add data
- Index data
- Manage search jobs
- Configure Splunk

For more information about the Splunk SDKs, read "Overview of the Splunk SDKs" in the Splunk Developer Portal.

Use Python SDK to export data

The Splunk SDK for Python lets you write Python applications that can interact with Splunk deployments. Export searches using the Python SDK can be run in historical mode and real-time mode. They start right away, and stream results instantly, letting you integrate them into your Python application.

Perform an export search using the Python SDK.

1. Set the parameters of what you wish to search. The following example sets the parameters as an export search of `splunklib` in the last hour.

```
import splunklib.client as client
import splunklib.results as results
```

2. Change or acquire these values, as necessary.

```
HOST = "localhost"
PORT = 8089
USERNAME = "admin"
PASSWORD = "changeme"
```

3. Run a normal-mode search.

```
service = client.connect(
    host=HOST,
    port=PORT,
```

```
    username=USERNAME,
    password=PASSWORD)

rr = results.ResultsReader(service.jobs.export("search index=_internal
earliest=-1h | head 5"))
```
4. Get the results and display them using the ResultsReader.

```
for result in rr:
    if isinstance(result, results.Message):
        # Diagnostic messages might be returned in the results
        print '%s: %s' % (result.type, result.message)
    elif isinstance(result, dict):
        # Normal events are returned as dicts
        print result
assert rr.is_preview == False
```

Use Java SDK to export data

The Java SDK is able to conduct and export searches while using Java.

To perform an export search using the Java SDK, run the following example in the `/splunk-sdk-java` directory using the CLI:

```
java -jar dist/examples/export.jar main --username="admin"
--password="changeme"
```
The Export application exports the "main" index to `export.out`, which is saved to the current working directory. If you want to run this application again, delete `export.out` before you try again. If you do not do this, you will get an error.

Here is a different CLI example of the Java SDK. It shows how to include a search query and change the output format to JSON.

```
java -jar dist/examples/export.jar main --search="search
sourcetype=access_*" json
```

Use JavaScript Export to export data

The Javascript Export endpoint can export Splunk data in the Javascript framework. Though the Splunk Javascript SDK does not currently support the Javascript Export endpoint, you can use a node javascript (.js) application request to export data.

To perform an export search using the Javascript Export endpoint:

1. Load the request module. Request is designed to be the simplest way to make an http/https call.

```
var request = require('request');
```

2. Call **get** to issue a GET request. Enter the following parameters:

- `strictSSL` ? When set to false, `strictSSL` tells the request to not validate the server certificate returned by your Splunk deployment, which by default is not a valid certificate.
- `uri` ? Provide the `uri` of the Splunk host along with the path for the export endpoint. A JSON response is specified in the query string.
- `qs` ? Set `qs` to supply the search parameter. By passing it this way, you do not have to URI encode the search string.

```
request.get(
    {
        strictSSL: false,
        uri:
'https://localhost:8089/servicesNS/admin/search/search/jobs/
              export?output_mode=json',
        qs: {
            search: 'search index=_internal'
        }
    }
)
```

3. Call `auth` to use HTTP Basic Auth and pass your Splunk username and password.

```
.auth('admin', 'changeme', false)
```

4. Pipe the results to `stdout`.

```
.pipe(process.stdout);
```

Use C# SDK to export data

An export search using the C# SDK runs asynchronously and immediately, does not create a job for the search, and starts streaming results right away. The C# SDK is useful when exporting large amounts of historical or real-time data.

To perform an export search using the C# SDK:

1. Create a preview search using StreamReader.

```
SearchPreviewStream searchPreviewStream;
```

2. Export the search result previews.

```
using (searchPreviewStream = service.ExportSearchPreviewsAsync("search
index=_internal | head 100").Result)
{
    int previewNumber = 0;
```

3. Enumerate through each search result preview.

```
    foreach (var searchPreview in searchPreviewStream.ToEnumerable())
    {
        Console.WriteLine("Preview {0:D8}: {1}", ++previewNumber,
searchPreview.IsFinal ? "final" : "partial");
        int recordNumber = 0;

        foreach (var result in searchPreview.Results)
        {
            Console.WriteLine(string.Format("{0:D8}: {1}",
++recordNumber, result));
        }
    }
}
```

Export data using the dump command

You can use the `dump` search command to export large collections of events onto a local disk. You can use this command with the CLI, Splunk SDK, and Splunk Web.

The basic syntax of the `dump` command is:

```
dump basefilename=<string> [rollsize=<number>] [compress=<number>]
[format=<string>] [fields=<comma-delimited-string>]
```

The `<format>` is the data format of the dump file that you are creating. Your format options are `raw`, `csv`, `tsv`,`xml`, and `json`.

For search examples and full explanations of the required and optional arguments, see the `dump` command in the *Search Reference*.

Forward data to third-party systems

Splunk software can forward data to third-party systems as follows:

- Through a plain TCP socket
- Packaged in a standard syslog

To forward data to third-party systems, you configure heavy forwarders by editing the `outputs.conf`, `props.conf` and `transforms.conf` files. This export method is similar to routing your data to other Splunk deployments. You can filter the data by host, source, or source type.

See Forward data to third party systems in the *Forwarding Data* manual.

Write Custom Search Commands

About writing custom search commands

You can extend the Splunk Search Processing Language (SPL) by customizing the built-in commands, or by writing your own search commands for custom processing or calculations.

If you use Splunk Cloud, you do not have filesystem access to your Splunk deployment. If you want to create custom search commands, file a Support ticket.

The following table describes the protocols, formats, and SDKs that you can use to create custom search commands.

Supported protocols	Description	Supported executable formats	SDK
Custom Search Command protocol, Version 2	Use to create custom commands for a wide range of platforms and executable formats.	.bat, .cmd, .exe, .js, .pl, .py, .sh	Splunk SDK for Python
Custom Search Command protocol, Version 1	Use with the Splunk SDK for Python to create custom commands for Python. Use with the Intersplunk.py SDK only to support existing custom commands.	.pl, .py	Splunk SDK for Python Intersplunk.py

Custom search commands that use Version 2 of the Custom Search Command protocol can be implemented in a variety of programming languages. These custom commands can even be implemented as platform-specific binary files.

By contrast, custom search commands that use the Version 1 protocol can be implemented only in Python. Custom commands that use the Version 1 protocol can only run using the Python interpreter that is included with the Splunk software.

About the SDKs

Use the Splunk SDK for Python to create custom search commands. The Splunk SDK for Python includes several templates that you can use to build new custom search commands.

Intersplunk.py is an older SDK and should only be used to support existing custom search commands that were built using the Version 1 protocol. You should not use the Intersplunk.py SDK for new custom search commands.

About the protocols

Version 2 protocol

There are significant advantages to using the Version 2 of the Custom Search Command protocol.

- With the Version 2 protocol, external programs process through an entire set of Splunk search results in one invocation. The external program is invoked once for the entire search, greatly reducing runtime overhead.
- The Version 2 protocol requires fewer configuration attributes than the Version 1 protocol.
- Supports non-Python custom search commands, for example C++, Go, Java and JavaScript (Node.js).
- Support for platform-specific executable files and binaries. You can write custom search commands in compiled languages, such as C++, on multiple platforms.

Version 1 protocol

The Version 1 of the Custom Search Command protocol processes events in groups, using 50,000 events for each group. The external program is invoked and terminated multiple times for large result sets.

See also

- Write a custom search command
- Security responsibilities with custom commands

Write a custom search command

A custom search command is an executable file that reads data in and writes data out. This could be a Python script, a C++ program, or some other executable binary file. For simplicity, the file is referred to as the **executable** in this documentation.

Search command executable requirements

The search command executable should be located in the appropriate directory and given a name that follows some basic rules.

Executable location
: The search command executable must be located in the appropriate `$SPLUNK_HOME/etc/apps/<app_name>/bin/` directory. Most of the executables that ship with Splunk Enterprise are associated with the Search & Reporting app and are stored in `$SPLUNK_HOME/etc/apps/search/bin`. Refer to executables in that directory for examples.

  If you use Splunk Cloud and want to create custom search commands, you must file a Support ticket. You do not have filesystem access to your Splunk Cloud deployment.

Executable name
: The executable name should be the same as the command name that you will invoke in your Splunk search and follow the naming convention `<command_name>.py`. Search command names can consist of only alphanumeric (a-z and 0-9) characters. New commands should have unique names. The name cannot be the same name of any of the built-in or existing custom commands.

Build the command executable using the Splunk SDK for Python

With the Version 2 protocol, you can use platform-specific versions of external search executables. See Select a location for your custom search command.

Note: For information about handling inputs, outputs, and errors with the Splunk SDK for Python, see How to create custom search commands using Splunk SDK for Python on the Splunk dev portal.

After you build the search command executable, you must Add the custom command to your Splunk deployment.

Build the command executable using the Intersplunk.py file

As part of building the executable, you need to specify how to handle inputs, send output, and handle errors.

Handling inputs

With the Intersplunk.py file, the input to the executable should be formatted in pure CSV or in Intersplunk, which is a header section followed by a blank line followed by pure CSV body.

The simplest way to interpret your executable input is to use `splunk.Intersplunk.readResults`, which takes three optional parameters and returns a list of dicts (which represents the list of input events). The optional parameters are `input_buf`, `settings`, and `has_header`.

inputbuf
: The `inputbuf` parameter where to read input from. If the parameter is None, which is the default value, the input is read from `sys.stdin`.

settings
: The `settings` parameter is expected to be a dict where any information found in the input header is stored. The default value for this parameter is None, which means that no settings are recorded

has_header
: The `has_header` parameter specifics whether or not input header is used. The default value for this parameter is True.

To indicate if your executable expects a header, use the `enableheader` attribute. The default value for the `enableheader` attribute is True, which means that the input will contain the header section and you are using the Intersplunk format.

If your executable does not expect a header section in the input, `enableheader` is False, you can directly use the Python CSV module to read the input. For example:

import csv

```
r = csv.reader(sys.stdin)
for l in r:
...
```

The advantage of using the Python CSV is that you can break at any time in the for loop, and only lines in the input that you had iterated to at that point are read into memory. This leads to much better performance in some use cases.

Sending output

You can also use the Intersplunk.py file to construct your output of the executable. `splunk.Intersplunk.generateErrorResults` takes a string and writes the correct error output to `sys.stdout`. `splunk.Intersplunk.outputResults` takes a list of dict objects and writes the appropriate CSV output to `sys.stdout`.

To output data, add:

```
splunk.Intersplunk.outputResults(results)
```

The output of your executable is expected to be pure CSV. For an error condition, simply return a CSV with a single "ERROR" column and a single row (besides the header row) with the contents of the message.

Handling errors

The arguments that are passed to your executable (in sys.argv) are the same arguments that are used to invoke your custom command in the search language, unless your executable has the attribute `supports_getinfo = true`. The `supports_getinfo` attribute indicates that the first argument to your executable is either `__GETINFO__ or __EXECUTE__`. This allows you to call the executable with the command arguments at parse time to check for syntax errors before any execution of the search. Errors at this time will short circuit any real execution of the search query. If called with `__GETINFO__`, this also allows you to dynamically specify the properties of your executable (such as streaming or not) depending on your arguments.

If your executable has the attribute `supports_getinfo` set to 'true', you should first make a call like:

```
(isgetinfo, sys.argv) = splunk.Intersplunk.isGetInfo(sys.argv)
```

This call will strip the first argument from sys.argv and check if you are in GETINFO mode or EXECUTE mode. If you are in GETINFO mode, your

executable should use `splunk.Intersplunk.outputInfo()` to return the properties of your executable or `splunk.Intersplunk.parseError()` if the arguments are invalid.

The definition of `outputInfo()` and its arguments is as follows:

def outputInfo(streaming, generating, retevs, reqsop, preop, timeorder=False)

You can also set these attributes in the commands.conf file. See How to edit a configuration file.

See also

- To decide where to implement your command, see Select a location for your custom search command.
- To learn about the supported protocols and SDKs, see About writing custom search commands.
- To learn about security best practices with custom search commands, see Security responsibilities with custom commands.
- For information about the custom search command parameters, see commands.conf.

Select a location for your custom search command

When you create a custom search command, you must update the `commands.conf` file in a **local** directory.

If you use Splunk Cloud, you do not have filesystem access to your Splunk Cloud deployment. You must file a Support ticket to add a custom search command to your deployment.

Locate the correct commands.conf file

The default directory, `$SPLUNK_HOME/etc/system/default`, contains preconfigured versions of the configuration files. **Never change or copy the configuration files in the default directory**. The files in the default directory must remain intact and in their original location.

Instead, you need to identify a local directory to put your custom search command in. Selecting the correct location is essential.

1. Determine the scope of the command.

Scope	Description
Application-specific custom command	Add application-specific commands to the `commands.conf` file in the local directory for the application. The location of an application local directory is `$SPLUNK_HOME/etc/apps/<app_name>/local`.
System-wide custom command	Add system-wide commands to the `commands.conf` file in local directory for the system. The location of the system local directory is `$SPLUNK_HOME/etc/system/local` .

2. Determine whether the `commands.conf` file already exists in your preferred local directory. If the file does not exist in the directory, create an empty `commands.conf` file in that directory. Do not copy the `commands.conf` file from the default directory.

Decide where to place the executable

You also need to determine where to place the custom command executable file. The Splunk software expects to find the executable file in all of the appropriate application directories. In most cases, you should place your executable file in an app namespace.

The following table shows where the executable file should be located, based on the location of the `commands.conf` file that contains the stanza for the custom command.

Commands.conf file location	Required script file location

`$SPLUNK_HOME/etc/apps/<app_name>/local`	`$SPLUNK_HOME/etc/apps/<app_name>/bin`
	
	If your command is platform-specific, the location is:
	`$SPLUNK_HOME/etc/apps/<app_name>/<PLATFORM>/bin/`

`$SPLUNK_HOME/etc/system/local`	`$SPLUNK_HOME/etc/system/bin`

There is one exception. To use an external process to run your executable file, you do not place your executable file in the `bin` directory in your apps. Instead, you must specify the executable location in a `.path` file. The `.path` file must be stored in one of the `bin` directories in your apps. See Using external programs to process command executables.

How the Splunk software finds your custom command

You register a custom search command by adding a stanza in the appropriate local `commands.conf` file.

For example, to add the custom command "fizbin" to your deployment, you would add the following stanza to the `commands.conf` file.

```
[fizbin]
chunked = true
```

Adding the stanza is described in detail in the topic Add the custom command to your Splunk deployment. However, you need to understand how the software locates your custom command executable before you actually add the stanza to the `commands.conf` file.

To find the executable to run your custom search command, the Splunk software searches in two places:

- The platform-specific application `bin` directory,

```
$SPLUNK_HOME/etc/apps/<app_name>/<PLATFORM>/bin/
```

- The default application `bin` directory,
  `$SPLUNK_HOME/etc/apps/<app_name>/bin/`

Platform-specific custom commands

The following table shows the supported platform-specific `bin` directories and the file extensions that are searched.

Platform architectures	Directory	File extensions
Linux on 64-bit x86_64	`linux_x86_64/bin`	.sh, .py, .js, and no extension
Linux on 32-bit x86	`linux_x86/bin`	.sh, .py, .js, and no extension
Mac OS X on 64-bit x86_64	`darwin_x86_64/bin`	.sh, .py, .js, and no extension

Windows on 64-bit x86_64	`windows_x86_64/bin`	.bat, .cmd, .py, .js, .exe
Windows on 64-bit x86_64	`windows_x86_64/bin`	.bat, .cmd, .py, .js, .exe

For example, when you use the `fizbin` command on a Linux 64-bit Splunk instance, the following paths are searched:

```
$SPLUNK_HOME/etc/apps/<app_name>/linux_x86_64/bin/fizbin.sh
$SPLUNK_HOME/etc/apps/<app_name>/linux_x86_64/bin/fizbin.py
$SPLUNK_HOME/etc/apps/<app_name>/linux_x86_64/bin/fizbin.js
$SPLUNK_HOME/etc/apps/<app_name>/linux_x86_64/bin/fizbin
$SPLUNK_HOME/etc/apps/<app_name>/bin/fizbin.sh
$SPLUNK_HOME/etc/apps/<app_name>/bin/fizbin.py
$SPLUNK_HOME/etc/apps/<app_name>/bin/fizbin.js
$SPLUNK_HOME/etc/apps/<app_name>/bin/fizbin
```

The Splunk software stops searching when a file with the same name as the command is found, in this example `fizbin`.

It is a good idea to include a platform-neutral version of your executable in the default application `bin` directory, `$SPLUNK_HOME/etc/apps/<app_name>/bin/`. This is useful if someone runs your custom command executable on a platform that you did not provide an implementation for.

You can also explicitly specify the executable that the Splunk software should look for by specifying the `filename` attribute in the `commands.conf` file. For example, assume the `fizbin` command is defined in the `commands.conf` file as follows:

```
[fizbin]
chunked = true
filename = fizbin.py
```

In this example, the Splunk software does not attempt to guess file extension. Instead, the software searches for the `fizbin.py` file only in the locations where a Python executable is expected.

```
$SPLUNK_HOME/etc/apps/<app_name>/linux_x86_64/bin/fizbin.py
$SPLUNK_HOME/etc/apps/<app_name>/bin/fizbin.py
```

Processing file extensions

When your custom command executable is located, the Splunk software looks for a file extension to determine how to run your command.

Filename extension	Action
`.py`	The Python interpreter `$SPLUNK_HOME/bin/python`, that is included with the Splunk software, is used to run your command.
`.js`	The Node.js runtime `$SPLUNK_HOME/bin/node`, that is included with the Splunk software, is used to run your command.
The executable file has no extension, or the file extension is not recognized	The Splunk software attempts to run the executable directly, without an interpreter. On UNIX-based platforms, this means that the executable must have the executable bit set.

Specifying command arguments

You specify command line arguments to use by adding `command.arg.<N>` attributes to the `commands.conf` file stanza. For example, if you want to pass a flag like `--verbose` to the `fizbin.py` executable, you add the following attributes in the `commands.conf` file stanza:

```
[fizbin]
chunked = true
filename = fizbin.py
command.arg.1 = --verbose
```

You can specify any number of `command.arg.<N>` arguments. For example:

```
[fizbin]
chunked = true
filename = java.path   #See the next section for filename examples
command.arg.1 = fizbin.jar
command.arg.2 = -classpath
command.arg.3 = <CLASSPATH>
```

The last segment of the argument must be a number. Arguments are sent for processing in numerical order. Any numbers that are skipped are ignored. Environment variables, such as `$SPLUNK_HOME`, are substituted in these arguments.

Using external programs to process command executables

Searches are processed one command at a time. The results of the previous command are sent to the next command. When the search reaches a custom

command, the search uses the protocol to send the results of the previous command to a separate process. The separate process can be a built-in process or an external process.

The Splunk software includes a Python interpreter and a JavaScript runtime environment. By default, if your custom command executable is a Python script or JavaScript file, the command executable is run on appropriate the executable processor that is included with the Splunk software.

If your executable is not a Python script or JavaScript file, or if you want to use a executable processor that is on your system, you must specify the location of the external program that you want to use to process your executable.

Java example

For example, you want to use a Java file to run the custom search. The Splunk software does not include a Java runtime environment (JRE). You need to specify the path to the JRE.

1. Create a `.path` file, such as `$SPLUNK_HOME/etc/apps/<app_name>/bin/java.path`. The `.path` file must be stored in one of the `bin` directories in your applications.

2. In the `.path` file, specify the path to the Java runtime environment (JRE). For example, `/usr/bin/java`.
3. In the `commands.conf` file, define your command by specifying the `filename` and the `command.arg.N` arguments. Absolute paths are not supported in the `filename` attribute. The following example shows the stanza for the `fizbin` command.

```
[fizbin]
chunked = true
filename = java.path
command.arg.1 = fizbin.jar
command.arg.2 = -classpath
command.arg.3 = <CLASSPATH>
```

   In this example, the Splunk software searches for the `java.path` file.

   Any environment variables that are specified, such as `$JAVA_HOME` are substituted in the `.path` file.

Python example

For example, you want to use a Python interpreter on your operating system instead of the Python interpreter that is included with the Splunk software.

1. Create a `.path` file, such as $SPLUNK_HOME/etc/apps/<app_name>/bin/system_python.path. The `.path` file must be stored in one of the `bin` directories in your apps.
2. In the `.path` file, specify the path to the Python interpreter. For example, `/usr/bin/python`.
3. In the `commands.conf` file, define your command by specifying the `filename` and `command.arg.1` attributes. Absolute paths are not supported in the `filename` attribute. The following example shows the stanza for the `fizbin` command.

```
[fizbin]
chunked = true
filename = system_python.path
command.arg.1 = fizbin.py
```

   In this example, the Splunk software searches for the `system_python.path` file.

Any environment variables that are specified, such as `$PYTHON_PATH` are substituted in `.path` file.

See also

Add the custom command to your Splunk deployment

Add the custom command to your Splunk deployment

You must add the custom command to the appropriate `commands.conf` configuration file.

Prerequisites

Review the following topics.

- Write a custom search command
- Select a location for your custom search command

If you use Splunk Cloud, you do not have filesystem access to your Splunk Cloud deployment. You must file a Support ticket to add a custom search command to your deployment.

The tasks to add a custom command to your deployment are:

1. Create or edit the `commands.conf` file in a **local** directory.
2. Add a new stanza to the `commands.conf` file that describes the command.
3. Restart Splunk Enterprise.

Add a new stanza to the local commands.conf file

Edit the local `commands.conf` file, to add a stanza for the command.

Each stanza in the `commands.conf` file represents the configuration for a specific search command. The following example shows a stanza that enables your custom command script:

[<stanza_name>]
chunked=true

filename = <string>

The `stanza_name` is the keyword that is used in searches to invoke the command. The `stanza_name` is also the name of the search command. Search command names must be lowercase and consist only of alphanumeric (a-z and 0-9) characters. Command names must be unique. The `stanza_name` cannot be the same as any other custom or built-in commands.

The `chunked=true` attribute specifies that the command uses the Version 2 protocol.

The `filename` attribute specifies the name of your custom command script. The `filename` attribute also specifies the location of the custom command script.

For example, to create the custom command "fizbin", you create a stanza in the `commands.conf` file.

```
[fizbin]
chunked = true
filename = fizbin.py
```

Other attributes that you can use to describe the custom command are explained later in this topic.

Describe the command (Version 2 protocol)

Version 2 of the Custom Search Command protocol dynamically determines if the command is a generating command, a streaming command, or a command that generates events.

Additionally, an authentication token is always sent to search commands that use the protocol.

The attributes that you can specify with the protocol are described in the following table.

Attribute	Description
command.arg.<N>	Additional command-line arguments to use when invoking the custom search command script. Environment variables such as $SPLUNK_HOME, are substituted.
filename	The name of the script to run when the custom search

	command is used.
is_risky	When users click a link or type a URL that loads a search into Splunk Web, if the search contains risky commands a warning appears. This warning does not appear when users create ad hoc searches. Specify this attribute if your custom search command is risky. Examples of build-in risky commands are `delete` and `dump`. To determine if your custom command is risky, see Safeguards for risky commands in the *Securing Splunk Enterprise* manual.
maxchunksize	The maximum size chunk, the size of metadata plus the size of the body, that the external command can produce. If the command tries to produce a chunk that is larger than the maxchunksiz value, the command is terminated.
maxwait	The maximum number of seconds the custom search command can pause before producing output.

Read more about these configuration attributes in the commands.conf.spec topic in the *Admin Manual*.

Describe the command (Version 1 protocol)

Some of the attributes you can use to describe your custom command using the Version 1 protocol specify the type of command.

1. You need to understand the differences between the types of commands. There are four broad categorizations for all the search commands:
   - Distributable streaming
   - Centralized streaming
   - Generating
   - Transforming

   For a comprehensive explanation about the command types, see Types of commands in this manual. For a complete list of the built-in commands that are in each of these types, see Command types in the *Search Reference*.
2. Describe the type of custom search command in the commands.conf file.
   1. Specify either the `streaming` or `generating` parameter in commands.conf file. Use these attributes to specify whether it is a generating command, a streaming command, or a command that generates events.
   2. You can also specify whether your custom command retains or transforms events with the `retainevents` parameter.

For a list of configurable settings, see the commands.conf reference in the *Admin Manual*. For example:

generating = [true|false|stream]

- Specify whether your command generates new events.
- If stream, then your command generates new events (generating = true) and is streamable (streaming = true).
- Defaults to false.

streaming = [true|false]

- Specify whether the command is streamable.
- Defaults to false.

If the custom search command retains or transforms events, include the `retainevents` attribute:

retainsevents = [true|false]

- Specify 'true' if the command retains events, similar to the `sort`, `dedup`, or `cluster` commands. Specify 'false' if the command transforms events, similar to the `stats` command.
- Default is false.

These are only a few of the attributes that you can specify in the stanza for your custom search command.

You can see the full list of configuration attributes in the commands.conf.spec topic in the *Admin Manual*.

Restart Splunk Enterprise

After you add the custom command to the appropriate `commands.conf` file, you must restart Splunk Enterprise.

Changes to your custom command script, or to the parameters of an existing command in the `commands.conf` file, do not require a restart.

See also

Control access to the custom command and script

Control access to the custom command and script

After you write the script and add it to `commands.conf`, you are good to go.

By default, all roles have read access to `commands.conf`, but only admins have write access. This means that all roles can run the commands listed in `commands.conf`, unless the access controls are explicitly changed for an individual command. If you want to restrict the usage of the command to certain roles or users, you must modify the access controls for the command.

Change custom command permissions

You can modify the access controls through the **Settings** menu, or by editing the `default.meta.conf` file.

Change permissions in Splunk Web

You can use the **Settings** menu to change the access controls for a command, by user role.

1. In Splunk Web, select **Settings**, **Advanced search**.

2. Click **Search commands**.

3. Under the **Sharing** column for the search command, click **Permissions**. This opens the **Permissions** view for the selected search command. Use this page to specify:

- If the command should appear in the current app or all apps.
- Which roles are have read and write access to the command.

4. Don't forget to save your changes!

Change permissions in the default.meta.conf file

You can change the access controls for a command using the `default.meta.conf` file, which is located in the

`$SPLUNK_HOME/etc/apps/<app_name>/metadata/` directory.

The following example shows the default access for the `commands.conf` and the access permissions for the input command, which you cannot run unless you are an admin.

```
[commands]
access = read : [ * ], write : [ admin ]
export = system

[commands/input]
access = read : [ admin ], write : [ admin ]
```

Change access control to the command script files

You can change the access control restrictions on the command script files. These controls are defined in the `[searchscripts]` stanza in the `$SPLUNK_HOME/etc/apps/<app_name>/metadata/default.meta` file. By default, the files are visible to all roles and apps, but only users with file system access, such as system administrators, can edit the files.

```
[searchscripts]
access = read : [ * ], write : [ admin ]
export = system
```

Use the `export = system` attribute to make files available to all apps in the system. In the examples above, access to `commands.conf` and `[searchscripts]` are global. If the global export under `[searchscripts]` is not present, the script configurations in the `commands.conf` file is visible in all apps, but the script files themselves are not.

Custom commands in apps that do not have a UI should be exported to the system, since there is no way to run the command in a local context.

Disable the custom command

You can use the **Settings** menu to disable a search command from running in an app

1. In Splunk Web, select **Settings**, **Advanced search**.

2. Click **Search commands**.

The Search commands page displays a table which lists the commands, information about the owner and app associated with the command, and provides options to restrict permissions and disable the command.

Note: This table only lists the search commands that are written in Python.

3. Under the **Status** column for the search command, click **Disable**.

A message banner towards the top of the window appears that confirms that the command was disabled in the app.

See Also

- Security responsibilities with custom commands
- default.meta.conf file in the *Admin manual*

Custom search command example

This topic applies to only the **Intersplunk.py file** and the Version 1 protocol.

For a Version 2 protocol example, see How to create custom search commands using Splunk SDK for Python on dev.splunk.com. There are several examples on that page:

- Starter example
- Basic example
- Shape example

Additionally there are other examples for the Splunk SDK for Python.

This following is an example of a custom search command called `shape`. The `shape` command categorizes events based on the event line count (tall or short) and line length (thin, wide, and very_wide) and whether or not the lines are indented.

Add the Python script

Add this script, `shape.py`, to an appropriate apps directory, `$SPLUNK_HOME/etc/apps/<app_name>/bin/` :

```
 import splunk.Intersplunk
  def getShape(text):
       description = []
       linecount = text.count("\n") + 1
       if linecount > 10:
           description.append("tall")
       elif linecount > 1:
           description.append("short")
       avglinelen = len(text) / linecount
       if avglinelen > 500:
           description.append("very_wide")
       elif avglinelen > 200:
           description.append("wide")
       elif avglinelen < 80:
           description.append("thin")
       if text.find("\n ") >= 0 or text.find("\n\t") >= 0:
           description.append("indented")
       if len(description) == 0:
           return "normal"
       return "_".join(description)
  # get the previous search results
  results,unused1,unused2 = splunk.Intersplunk.getOrganizedResults()
  # for each results, add a 'shape' attribute, calculated from the raw
event text
  for result in results:
       result["shape"] = getShape(result["_raw"])
  # output results
  splunk.Intersplunk.outputResults(results)
```

Edit the configuration files

Edit the following configuration files in the local directory for the app, for example `$SPLUNK_HOME/etc/app/<app_name>/local`.

1. In the `commands.conf` file, add this stanza:
   [shape]
   filename = shape.py
2. In the `authorize.conf` file, add these two stanzas:
   [capability::run_script_shape]
   [role_admin]
   run_script_shape= enabled
3. Restart Splunk Enterprise.

Run the command

This example shows how to run the search from the CLI. You can also run the command in Splunk Web.

Show the top *shapes* for multi-line events:

$ splunk search "linecount>1 | shape | top shape"

The results of the search are returned in a table format.

```
shape                       count     percent

tall_indented               43        43.000000
short_indented              29        29.000000
tall_thin_indented          15        15.000000
short_thin_indented         10        10.000000
short_thin                  3         3.000000
```

Security responsibilities with custom commands

As the author of a custom search command, it is your responsibility to follow best security practices when developing custom search commands. This topic includes information about writing high-quality, secure custom search commands.

Custom search commands run with the same permissions as splunkd on the search heads and indexers of a Splunk software deployment. Custom search commands do not run in a sandbox. Consequently, the security of your deployments depends on the security of your custom search commands.

Validate search results

First and foremost, you should consider data coming from search results as untrusted user input. Search results might contain arbitrary strings. If you use these strings unescaped or without validation within your program, you might unwittingly introduce critical security vulnerabilities.

For example, if you pass an unvalidated field from a search result as the argument to a shell command, unescaped semi-colons might allow malicious individuals to run arbitrary programs on a Splunk deployment. It is your responsibility as a search command author to validate input and avoid code injection and path traversal vulnerabilities, for example:

- Code injection vulnerabilities, such as SQL injection
- Path traversal vulnerabilities, such as allowing user data to reference arbitrary file system paths

Use role-based access control

By default, custom search commands are available to any user who is logged in. It is a good practice to restrict access search commands by user roles.

Some search commands might be written to perform privileged maintenance actions, such as a command to "purge the database cache". Only users with special privileges should be allowed to use these commands.

You can use the role-based access control features to restrict permission to run search commands. You can restrict permissions to specific users or roles.

For example, you have a custom search command called "launchmissiles" that you want only users assigned to the "admin" role to be able to use. In this situation, you would follow these steps:

1. Create a `default.meta` file in the `metadata` directory in your application.
2. In the `default.meta` file, add the following content:

   ```
   [commands/launchmissiles]
   access = read : [ admin ], write : [ admin ]
   ```

   The `read` access specifies who can run a custom search command. The above example limits `read` access to the `admin` role. You can specify access to other roles, such as the "power" role, or a new role that you define in the `authorize.conf` file.

   Set `write` access only to the "admin" role. No other role should have `write` permissions.

See About users and roles in *Securing Splunk Enterprise*.

Avoid using the local file system

Avoid accessing or modifying the file system as much as possible. Any code that you write which accesses the file system might contain bugs that allow malicious individuals to exfiltrate data from a Splunk deployment. A bug could allow a user to read data from indexes that the individuals are not permitted to search.

Unconstrained file system access might lead to severe challenges when trying to deploy a custom search command in a search head cluster environment. For example, if you edit a `.conf` file directly in your custom search command, those changes would not be replicated to the other members of a search head cluster. You should always use the Splunk REST API to interact with `.conf` files or other

knowledge objects. If you need to permanently persist data from a custom search command, use the REST API to write to the KV store or to `.conf` files.

Temporary directories

If you must use the file system, for example to spill data from memory to disk, create a temporary directory under the `dispatch` directory for the search. Use a secure directory creation method, such as the `tempfile.mkdtemp()` function in Python. The `dispatch` directory is automatically purged by the Splunk software after a search expires.

- For custom search commands that use the Version 2 protocol, the dispatch directory is in the `dispatch_dir` attribute of the `searchinfo` JSON object that is sent with the `getinfo` action from the Splunk software.
- Do not use `/tmp`, `$TMPDIR`, or any similar strategy to find a temporary directory to write files to. Remember that the maximum length of a path on Windows is 260 characters.

Search Examples and Walkthroughs

What's in this section?

This section contains walkthroughs of interesting search examples collected from Answers and the field.

- Add comments to a search
- Calculate sizes of dynamic fields

Add comments to a search

The most flexible way to add comments to your search strings is to use the built-in comment macro. You can use the macro multiple times in your search string and multiple times in a single command string. Comments in a search do not impact search performance.

By default the comment macro is shared only in the Search app.

Using the comment macro

You can use the comment macro to add comments anywhere in your search string. The syntax for a comment is `` `comment("comment text")` ``.

Examples

```
`comment("THIS IS A COMMENT")`

`comment("This part of the search returns only one value")`
```

Comments begin and end with the back quote, or grave accent, character.

Adding multiple comments to a search

The following search example classifies recent earthquakes based on their depth.

```
source=usgs
| eval Description=case(depth<=70, "Shallow", depth>70 AND depth<=300,
```

```
 "Mid",
  depth>300, "Deep")
| stats count min(mag) max(mag) BY Description
```

When you add inline comments the search is easier to understand. This is the same search with multiple comments added to explain each part of the search.

```
source=usgs `comment("source is the us geological service (usgs)")`
| eval Description=case(depth<=70, "Shallow", depth>70 AND depth<=300,
"Mid",
  depth>300, "Deep")
  `comment("Creates field Description. Case function specifies
earthquake
  depths, returns Description values - Shallow, Mid, Deep.")`
| stats count min(mag) max(mag) `comment("Counts earthquakes, displays
min
  and max magnitudes")` BY Description
```

Consider using uppercase characters for your comments to make them easier to find. This is the same search with the comments in uppercase.

```
source=usgs `comment("SOURCE IS THE US GEOLOGICAL SERVICE (USGS)")`
| eval Description=case(depth<=70, "Shallow", depth>70 AND depth<=300,
"Mid",
  depth>300, "Deep")
  `comment("CREATES FIELD DESCRIPTION. CASE FUNCTION SPECIFIES
EARTHQUAKE DEPTHS, RETURNS DESCRIPTION VALUES - SHALLOW, MID, DEEP.")`
| stats count min(mag) max(mag) `comment("COUNTS EARTHQUAKES, DISPLAYS
MIN AND MAX MAGNITUDES")` BY Description
```

Using comments to troubleshoot a search

The following search example is attempting to return the bytes for the individual indexes. However, the search has the wrong field in the `stats` command <split-by clause>.

```
index=_internal source=*license* type=usage | stats sum(b) BY index
```

You can comment out portions of your search to help identify problems. Another option is to run the search in Verbose mode. In this search the `stats` portion of the search is commented out.

```
index=_internal source=*license* type=usage `comment("| stats sum(b) BY
index")`
```

The results show the correct name for the field. You need to specify **idx** as the field name instead of **index**.

```
index=_internal source=*license* type=usage | stats sum(b) BY idx
```

(Thanks to Splunk user Runals for this example.)

Calculate sizes of dynamic fields

This search determines which fields in your events consume the most disk space, without any prior knowledge of field names and number of events.

Scenario

```
index=_internal earliest=-15m latest=now
| fieldsummary
| rex field=values max_match=0 "value\":\"(?<values>[^\"]*)\","
| mvexpand values
| eval bytes=len(values)
| rex field=field
"^(?!date|punct|host|hostip|index|linecount|source|sourcetype|timeendpos|timestartpos|s
| stats count sum(bytes) as SumOfBytesInField values(values) as Values
max(bytes) as MaxFieldLengthInBytes by FieldName
| rename count as NumOfValuesPerField
| eventstats sum(NumOfValuesPerField) as TotalEvents
sum(SumOfBytesInField) as TotalBytes
| eval PercentOfTotalEvents=round(NumOfValuesPerField/TotalEvents*100,2)
| eval PercentOfTotalBytes=round(SumOfBytesInField/TotalBytes*100,2)
| eval ConsumedMB=SumOfBytesInField/1024/1024
| eval TotalMB=TotalBytes/1024/1024
| table FieldName NumOfValuesPerField SumOfBytesInField ConsumedMB
PercentOfTotalBytes PercentOfTotalEvents
| addcoltotals labelfield=FieldName label=Totals
| sort - PercentOfTotalEvents
```

The results appear on the Statistics tab and look something like this:

FieldName	NumValuesPerField	SumOfBytesInField	ConsumedMB	PercentOfTotalByt
Totals	1802	45700	0.0436	99.87
cumulative_hits	100	587	0.0006	1.28
eps	100	1862	0.0018	4.07
kb	100	1159	0.0011	2.54

kbps	100	1881	0.0018	4.12
req_time	100	3000	0.0029	6.56
uri	100	10559	0.0101	23.11
uri_query	100	3532	0.0034	7.73
message	96	11633	0.0111	25.46
avg_age	76	280	0.0012	2.80
ev	62	140	0.0001	0.31
average_kbps	59	1071	0.0010	2.34

Walkthough

Let's walk through each part of the search.

1. The example begins with a search to retrieve all events in `index=_internal` within the last 15 minutes.
   ```
   index=_internal earliest=-15m latest=now
   ```

   Note: You can replace this with any search string and time range.
2. Next, add the the fieldsummary command to create a summary of all the fields in the previously retrieved events.
   ```
   | fieldsummary
   ```

   The results appear on the Statistics tab and look something like this:

field	count	distinct _count	is _exact	max	mean	min	numeric _count	stdev	
abandoned _channels	29	1	1	0.0	0.00	0.0	29	0.00	[{"value":"0","cou
active	29	1	1	0.0	0.00	0.0	29	0.00	[{"value":"0","cou
active _hist _searches	31	2	1	1.0	0.13	0.0	31	0.34	[{"value":"0","cou {"value":"1","coun
average _kbps	87	59	1	0.3	0.21	0.0	87	0.15	[{"value":"0","cou {"value":"0.31239 {"value":"0.31240 {"value":"0.31245

									{"value":"0.31247

3. The values of each field are extracted with a regex into a multivalue field called **values**, and then expanded. The length of each value is calculated in bytes.

```
| rex field=values max_match=0 "value\":\"(?<values>[^\"]*)\","
| mvexpand values
| eval bytes=len(values)
```

4. The values of the field are extracted with another regex, with some exceptions.

```
| rex field=field
"^(?!date|punct|host|hostip|index|linecount|source|sourcetype|timeendpos|timestar
```

5. The `stats` command is used to perform multiple calculations using stats functions, including the count and the sum of the bytes (SumOfBytesInField). The values function is used to returns the list of all distinct values of the `values` field as a multivalue entry (Values). The max function calculates the maximum field length in bytes (MaxFieldLengthInBytes). The results are organized by field name.

```
| stats count sum(bytes) as SumOfBytesInField values(values) as
Values max(bytes) as MaxFieldLengthInBytes by FieldName
| rename count as NumOfValuesPerField
```

6. The `eventstats` command is used to calculate several sums, the number of values in each field (TotalEvents) and the sum of the bytes in each field (Total Bytes).

```
| eventstats sum(NumOfValuesPerField) as TotalEvents
sum(SumOfBytesInField) as TotalBytes
```

7. Several `eval` commands are run to calculate the percentage of total events, the percentage of total bytes, the megabytes consumed, and the total megabytes.

```
| eval
PercentOfTotalEvents=round(NumOfValuesPerField/TotalEvents*100,2)
| eval
PercentOfTotalBytes=round(SumOfBytesInField/TotalBytes*100,2)
| eval ConsumedMB=SumOfBytesInField/1024/1024
| eval TotalMB=TotalBytes/1024/1024
```

8. The `table` command is used to display on a specific set of fields. The `addfoltotals` command is used to calculate the total for each column. The `sort` command is used sort the list in descending order by the PercentageOfTotalEvents field.

```
| table FieldName NumberOfValuesPerField SumOfBytesInField
ConsumedMB PercentageOfTotalBytes PercentageOfTotalEvents
```

```
| addcoltotals labelfield=FieldName label=Totals
| sort - PercentageOfTotalEvents
```

The results appear on the Statistics tab and look something like this:

FieldName	NumValuesPerField	SumOfBytesInField	ConsumedMB	PercentOfTotalByt
Totals	1802	45700	0.0436	99.87
cumulative_hits	100	587	0.0006	1.28
eps	100	1862	0.0018	4.07
kb	100	1159	0.0011	2.54
kbps	100	1881	0.0018	4.12
req_time	100	3000	0.0029	6.56
uri	100	10559	0.0101	23.11
uri_query	100	3532	0.0034	7.73
message	96	11633	0.0111	25.46
avg_age	76	280	0.0012	2.80
ev	62	140	0.0001	0.31
average_kbps	59	1071	0.0010	2.34