



# **Splunk® Enterprise**

## **Developing Views and Apps for Splunk Web 9.0.1**

Generated: 8/24/2022 2:16 am

# Table of Contents

<b>Getting started.....</b>	<b>1</b>
Building customizations for the Splunk platform.....	1
<b>Custom visualizations.....</b>	<b>2</b>
Custom visualizations overview.....	2
API updates and migration advice.....	3
Build a custom visualization.....	6
Custom visualization API reference.....	19
Formatter API reference.....	40
Data handling guidelines.....	46
Design guidelines.....	47
Custom visualizations in Simple XML.....	55
Custom visualizations in SplunkJS.....	56
<b>Custom alert actions.....</b>	<b>58</b>
Custom alert actions overview.....	58
Custom alert action component reference.....	58
Set up custom alert configuration files.....	60
Create a custom alert action script.....	63
Define a custom alert action user interface.....	66
Optional custom alert action components.....	72
Convert a script alert action to a custom alert action.....	73
Logger example for custom alert actions.....	79
HipChat example for custom alert actions.....	81
Advanced options for working with custom alert actions.....	86
KV Store integration for custom alert actions.....	88
<b>Modular inputs.....</b>	<b>90</b>
Modular inputs basic example.....	90
Create modular inputs.....	95
Set up logging.....	106
Set up external validation.....	107
Data checkpoints.....	109
Set up streaming.....	111
Modular inputs configuration.....	116
Create a custom user interface.....	121
Developer tools for modular inputs.....	127
Modular inputs examples.....	132
<b>Build scripted inputs.....</b>	<b>148</b>
Setting up a scripted input.....	148
Writing reliable scripts.....	150
Example script that polls a database.....	154
<b>Customize Splunk Web.....</b>	<b>158</b>
Customization options and caching.....	158
Customize the login page.....	158

# Table of Contents

## Customize Splunk Web

Customize dashboard styling and behavior.....	161
UI internationalization.....	162

## Building custom apps.....165

Developer resources.....	165
--------------------------	-----

# Getting started

## Building customizations for the Splunk platform

Learn about APIs and other Splunk platform customization options.

### Developer resources

#### *APIs*

Learn how to build custom solutions for data input, analysis, and alert actions.

API	Use case
<a href="#">Custom visualizations</a>	Create custom visualizations for analyzing data patterns and trends.
Custom data inputs (modular inputs, scripted inputs)	Index data from unique sources or in non-standard formats.

#### *Splunk Web customization*

Modify login page and other components.

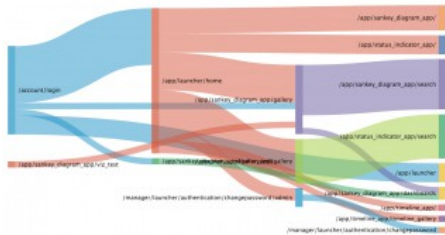
#### *See also*

See the following resources for additional information.

To learn about	See
<ul style="list-style-type: none"><li>• Building visualizations and dashboards</li><li>• Using Simple XML</li></ul>	<ul style="list-style-type: none"><li>• <i>Dashboards and Visualizations</i></li></ul>
<ul style="list-style-type: none"><li>• Working with alerts</li></ul>	<ul style="list-style-type: none"><li>• <i>Alerting manual</i></li></ul>
<ul style="list-style-type: none"><li>• General app building guidance</li><li>• Leveraging Splunk SDKs</li></ul>	<ul style="list-style-type: none"><li>• Develop apps and add-ons for Splunk Enterprise on the Splunk Developer Portal</li><li>• Splunk Enterprise SDKs on the Splunk Developer Portal</li></ul>
<ul style="list-style-type: none"><li>• Creating custom search commands</li><li>• Managing access to custom search commands</li><li>• Custom search command examples</li></ul>	<ul style="list-style-type: none"><li>• Create custom search commands on the Splunk Developer Portal</li></ul>
<ul style="list-style-type: none"><li>• Using the Splunk REST API</li></ul>	<ul style="list-style-type: none"><li>• <i>REST API Reference Manual</i></li><li>• <i>REST API User Manual</i></li><li>• <i>REST API Tutorials</i></li></ul>

# Custom visualizations

## Custom visualizations overview



Unique use cases and data sets can require custom visualizations.

Use the Splunk platform custom visualization API to create visualizations that admins can download and install from Splunkbase. Users can access and configure installed custom visualizations in Splunk Web. The API lets you create a user experience consistent with the standard Splunk platform visualization set.

For user documentation on Splunk platform custom visualization apps, see [Custom visualizations](#).

---

## Release notes

The custom visualization API has been updated for the latest Splunk software version. If you are building a new custom visualization app, use the latest version of the API.

Developers who built apps using prior versions of the API are encouraged to update their apps.

For more details on API updates and migration advice for existing custom visualization apps, see [API updates and migration advice](#).

---

## Developer resources

### Documentation

#### Tutorial

##### [Build a custom visualization](#)

Learn how to create a working custom visualization. This tutorial includes steps for creating an example visualization, developer best practices, and example code.

#### API details and best practices

##### [Custom visualization API reference](#)

Review custom visualization components and app directory structure.

##### [API updates and migration advice](#)

Get migration information for apps built using the previous API version.

## Formatter API reference

Review components of the user interface for formatting visualizations.

### User experience

#### Design guidelines

Implement custom visualization appearance and behavior.

#### Data handling guidelines

Work with user search results and data format errors.

### API interactions

#### Custom visualizations in Simple XML

Add a custom visualization to a dashboard and configure it in Simple XML.

#### Custom visualizations in SplunkJS

Access and instantiate a custom visualization in SplunkJS.

## Community

Discuss custom visualizations with other developers on Splunk Answers.

## API updates and migration advice

The custom visualizations API has been updated for the latest Splunk Enterprise software version.

If you used one of the previous versions of the API to build a custom visualization app, review the following versioned changes for details and migration requirements.

Changes to existing functionality require migration to support compatibility with the custom visualizations framework. Other updates add new functionality and migration is optional.

The Splunk Custom Visualization API does not support the base (search id="baseSearch" or post-process (search base="baseSearch") searches. Change these searches to inline searches to make them compatible. For more information see Overview of search types and uses in the Splunk Dashboards and Visualizations Manual.

## Backwards compatibility

App code migrated to the newest API version is no longer compatible with the previous Splunk Enterprise software version. For published apps, use versioning to support separate compatibility with different Splunk Enterprise versions.

## Migrating to 6.6

### Changes to existing functionality

Software version 6.6 introduces a drilldown editor user interface for visualizations saved to dashboards.

The drilldown editor is available in dashboard edit mode. As of software version 6.6, the visualization **Format** menu no longer includes drilldown enablement and configuration settings for Splunk platform visualizations.

After saving a visualization to a dashboard, users can use the drilldown editor or Simple XML to configure drilldown behavior.

Update	Type of update	Changes	How to migrate
<code>supports_drilldown</code>  Support flag for drilldown editor availability in <code>visualizations.conf</code>	A new UI editor is available in Splunk Enterprise 6.6. This flag determines whether the editor is available for a visualization.	Previous drilldown default behavior and configuration options in the Format menu might have changed.	Remove any existing drilldown configuration options in the Format menu. Add the <code>supports_drilldown</code> flag to <code>visualizations.conf</code> and set it to <code>true</code> to provide the drilldown editor option in dashboards.
<code>display.visualizations.custom.drilldown</code>  Drilldown enablement setting in the custom visualization. Located in <code>savedsearches.conf</code> .	Indicates whether drilldown is enabled by default. Default value is <code>all</code> (enabled).	Replaces prior drilldown enablement settings in the Format editor.	Remove any existing drilldown configuration options in the Format menu and use the <code>supports_drilldown</code> flag to make the drilldown editor available to users.  No other migration change is necessary.

### New functionality in 6.6

Software version 6.6 introduces trellis layout. This feature lets users split search results on a field or aggregation to visualize values in separate segments.

Update	Type of update	Changes	Replaces	How to migrate
<code>supports_trellis</code> Support flag for trellis layout in <code>visualizations.conf</code>	Trellis layout lets users split search results over a field or aggregation. Visualizations appear in segments to show each field value.	Adds a new UI option and menu for configuring trellis layout on visualizations.	N/A	Add the flag to <code>visualizations.conf</code> to make the trellis layout configuration menu available for the visualization.

## Migrating to 6.5

### Changes to existing functionality

The following updates for software version 6.5 require migration in any apps built using the previous API version.

Update	Where	Type of update	Changes	Replaces	How to migrate
<code>&lt;splunk-control-group&gt;</code>	<code>formatter.html</code>	Change to supported API	Provides a component for wrapping input controls with layout, labels, and help text.	Replaces CSS layout classes for formatter controls.  Replaces separate label	Use this component to wrap <code>formatter.html</code> elements instead of using <code>&lt;div&gt;</code> elements with CSS classes.

Update	Where	Type of update	Changes	Replaces	How to migrate
			CSS layout classes are deprecated and should no longer be used.	and help text <code>&lt;div&gt;</code> elements.	Labels and help text should no longer be declared explicitly using a separate <code>&lt;div&gt;</code> .  See the <a href="#">Formatter API reference</a> for more details.
Visualization namespaces syntax	<code>formatter.html</code>	Change to supported API	Provides an abbreviated syntax for referencing a visualization in the HTML file.	Previous fully qualified namespaces syntax should not be used.	Use the new namespaces for accuracy and stability.  See the <a href="#">Formatter API reference</a> for more details.
Base visualization interface and utility class location	<code>SplunkVisualizationBase</code> <code>SplunkVisualizationUtils</code>	File path change	The base interface and utility class files are located in the <code>api/</code> directory.	These files are no longer located in the <code>vizapi/</code> directory.	Update any file location references in your visualization code to use the <code>api/</code> directory.

### New functionality in 6.5

The following updates provide new functionality in the custom visualizations framework for software version 6.5. Migration is optional but recommended.

Update	Where	Changes	Replaces	How to migrate
New utility functions	<code>SplunkVisualizationUtils</code>	Provides new utilities for accessing color palettes, formatting dates and time, and boolean normalization.	N/A	Use the new utility functions to access commonly needed resources or functionality.  Continue using the required security utilities.  See <a href="#">Utility functions</a> in the API reference for more details.
<code>&lt;splunk-color-picker&gt;</code>	<code>formatter.html</code>	Provides a way to set a preconfigured color palette type, extend a preconfigured palette, or create a custom color palette.	N/A	Use one of the preconfigured color palettes or specify custom colors for extending and creating new palettes.  See the <a href="#">Formatter API reference</a> for more details.
<code>data.meta.done</code> flag indicating search completion.	<code>visualization.js</code>	Provides a way to check for search completion.	N/A	Use the flag to handle large results sets from long running searches.  See <a href="#">Data handling</a>



Update	Where	Changes	Replaces	How to migrate
				<a href="#">guidelines</a> for more details.

## Build a custom visualization

This tutorial introduces custom visualization app components and developer best practices.

For complete component information, see the [Custom visualization API reference](#).

### Tutorial overview

As an example project, the tutorial shows you how to build a radial meter visualization.



This custom visualization uses the D3.js library to render the radial meter. The meter visualization represents a search result count value as a partial circle on a value range.

## Getting started

### Development mode settings

It is not necessary for the Splunk platform to be in development mode while building a custom visualization. However, development mode provides access to unbuilt file content and prevents caching. These options are helpful for debugging as well as for viewing changes as you make them.

You can enable development mode by adding the following settings to the `web.conf` configuration file in `etc/system/local`. If you do not already have a local copy of this file, create one and add these settings.

```
[settings]
minify_js = False
minify_css = False
js_no_cache = True
cacheEntriesLimit = 0
cacheBytesLimit = 0
enableWebDebug = True
```

## Build the app

Start by setting up an app to contain the custom visualization. Use one of the following options.

App setup option	What to do	Notes
Download the app template	1. Download the custom visualization app template.	The template contains the directory and file structure for the app.

App setup option	What to do	Notes
	<p>2. Unzip the template in the <code>\$SPLUNK_HOME/share/splunk/app_templates/</code> directory.</p> <p>3. Place the finished app in the <code>\$SPLUNK_HOME/etc/apps</code> directory.</p>	
Create the app manually	Use the directory and file structure shown below.	<code>viz_tutorial_app</code> is used as the app name in this tutorial.
Add the visualization to an existing app	Add the relevant visualization files to the app directory. Use the directory and file structure shown below.	<code>viz_tutorial_app</code> is used as the app name in this tutorial.

### App directory and file structure

Here is the directory structure for an app that contains a custom visualization. The directory includes Webpack configuration files. Throughout this tutorial, `viz_tutorial_app` is used as the app name.

```

appname
  appserver
    static
      visualizations
        <visualization_name>
          src
            visualization_source.js
            webpack.config.js
            visualization.js
            visualization.css
            formatter.html
            package.json
            preview.png
          default
            visualizations.conf
            savedsearches.conf
          metadata
            default.meta
          README
            savedsearches.conf.spec

```

### Visualization files in the app

There are four main files in a custom visualization.

File	Description
<code>visualization_source.js</code>	This file contains the source code for the custom visualization. You can edit the source code in this file. Webpack uses the source code file to build the <code>visualization.js</code> file.
<code>visualization.js</code>	Built file for rendering the visualization.
<code>formatter.html</code>	Contains HTML for rendering the visualization format menu. The format menu shows up on the <b>Search</b> page and in dashboards.
<code>visualization.css</code>	Contains CSS style and behavior rules for the visualization. CSS rules should have names as specific to the visualization as possible.

## Additional components for the tutorial visualization

The radial meter visualization uses the D3 library to render the meter. The tutorial shows you how to use the `npm` package manager to install D3 in the app package. Webpack then builds the D3 library into the visualization code.

---

## Create the visualization logic

### *Set up the visualization source code*

Create visualization source code using the custom visualizations app template.

1. Rename the `viz_tutorial_app/appserver/static/visualizations/standin` directory to `radial_meter`.
2. Install dependencies using `npm` package manager.  
From the `viz_tutorial_app/appserver/static/visualizations/radial_meter` directory, run `$ npm install`.  
Disregard any warning messages that might appear.  
This step generates a `/node_modules` sub-directory in this folder.
3. Add D3 as a dependency.  
Run `$ npm install --save d3@3.5`.  
Observe that, for purposes of this tutorial, a specific version of D3 must be used.  
This step generates a `D3` directory in `/node_modules`.
4. Add other dependencies.
  - a.) Run `$ npm install --save underscore`.
  - b.) Run `$ npm install --save jquery`.
5. In the `viz_tutorial_app/appserver/static/visualizations/radial_meter/src` directory, find the `visualization_source.js` file. Replace all of the code in `visualization_source.js` with the following code.

### visualization\_source.js code

```
define([
  'jquery',
  'underscore',
  'api/SplunkVisualizationBase',
  'api/SplunkVisualizationUtils',
  'd3'
],
function(
  $,
  _,
  SplunkVisualizationBase,
  SplunkVisualizationUtils,
  d3
) {

return SplunkVisualizationBase.extend({

  initialize: function() {
    // Save this.$el for convenience
    this.$el = $(this.el);
```

```

        // Add a css selector class
        this.$el.addClass('splunk-radial-meter');
    },

    getInitialDataParams: function() {
        return ({
            outputMode: SplunkVisualizationBase.ROW_MAJOR_OUTPUT_MODE,
            count: 10000
        });
    },

    updateView: function(data, config) {

        // Fill in this part in the next steps.

    }
    });
});

```

---

### ***How visualization\_source.js manages visualization logic***

To manage visualization logic, `visualization_source.js` uses important custom visualization framework conventions.

It returns an object that extends the `SplunkVisualizationBase` class. As part of extending this class, `visualization_source.js` overrides two functions in `SplunkVisualizationBase`.

- `updateView`
  - This function is called whenever search results are updated or the visualization format changes. It handles visualization rendering using the following two parameters.
    - ◆ `data`. An object containing search result data.
    - ◆ `config`. An object containing visualization format information.
  - The next part of the tutorial shows you how to complete the `updateView` function.
- `getInitialDataParams`
  - This function is required for data to be returned from the search. This function specifies the data output format for search results. You can also use `getInitialDataParams` to specify the maximum number of results.

---

### ***Add the updateView function***

`updateView` needs to check for data and render the visualization according to the specified configuration. Add the following code to fill in the function in `visualization_source.js`. Read the inline comments to learn more about each part of the function.

#### **updateView code**

```

updateView: function(data, config) {

    // Guard for empty data
    if(data.rows.length < 1){
        return;
    }
    // Take the first data point
    datum = data.rows[0][0];
    // Clear the div

```

```

this.$el.empty();

// Pick a color for now
var mainColor = 'yellow';
// Set domain max
var maxValue = 100;

// Set height and width
var height = 220;
var width = 220;

// Create a radial scale representing part of a circle
var scale = d3.scale.linear()
    .domain([0, maxValue])
    .range([ - Math.PI * .75, Math.PI * .75])
    .clamp(true);

// Create parameterized arc definition
var arc = d3.svg.arc()
    .startAngle(function(d) {
        return scale(0);
    })
    .endAngle(function(d) {
        return scale(d);
    })
    .innerRadius(70)
    .outerRadius(85);

// SVG setup
var svg = d3.select(this.el).append('svg')
    .attr('width', width)
    .attr('height', height)
    .style('background', 'white')
    .append('g')
    .attr('transform', 'translate(' + width / 2 + ', ' + height / 2 + ')');

// Background arc
svg.append('path')
    .datum(maxValue)
    .attr('d', arc)
    .style('fill', 'lightgray');

// Fill arc
svg.append('path')
    .datum(datum)
    .attr('d', arc)
    .style('fill', mainColor);

// Text
svg.append('text')
    .datum(datum)
    .attr('class', 'meter-center-text')
    .style('text-anchor', 'middle')
    .style('fill', mainColor)
    .text(function(d) {
        return parseFloat(d);
    })
    .attr('transform', 'translate(' + 0 + ', ' + 20 + ')');
}

```

## Add data chunking to the `updateView` function

The Splunk custom visualization API only supports the return of 50,000 results at a time. If you're running searches that return over 50,000 results, you must use code that manages the data in chunks of 50,000 events at a time. To do this, add the following code in the `updateView` function above:

```
updateView: function(data, config) {
  // get data
  var dataRows = data.rows;

  // check for data
  if (!dataRows || dataRows.length === 0 || dataRows[0].length === 0) {
    return this;
  }

  // the rest of your updateView logic here

  // fetch the next chunk after processing the current chunk
  this.offset += dataRows.length;
  this.updateDataParams({count: this.chunk, offset: this.offset});
}
```

And add the following code to your `initialize` function:

```
initialize: function() {
  this.chunk = 50000;
  this.offset = 0;

  // the rest of your initialization logic here
}
```

To learn more about the `initialize` function, see: ["Interface Methods" in the Custom visualization API reference section](#).

---

## Add CSS

At this point, you can add CSS rules to manage the visualization appearance. Add the following rules to the `visualization.css` file.

```
/* Formatting for text element*/
.meter-center-text {
  font-size: 40px;
  font-weight: 200;
  font-family: "Helvetica Neue", Helvetica, sans-serif;
}

/* Center the main SVG in the page */
.splunk-radial-meter svg {
  display: block;
  margin: auto;
}
```

---

## Add configuration settings

Register and export the visualization using configuration files in the app directory.

## Register the visualization

Register the visualization to make it visible to the Splunk platform.

1. In the `viz_tutorial_app/default` folder, find the `visualizations.conf` file.
2. Open the file and delete the entire `[standin]` stanza.
3. Add the following stanza to the file. Note that the stanza name must match the visualization folder name. In this case, it is `radial_meter`.

```
[radial_meter]
label = Radial Meter
```

Every visualization in an app must have a stanza in the `visualizations.conf` file. **The stanza name and the `label` attribute are required** but there are other optional settings. Here is the complete list of settings that the stanza can include.

Settings	Description	Required?
<code>label</code>	Public label used throughout Splunk Web to refer to the visualization.	<b>Yes</b>
<code>default_height</code>	Default visualization height.	No. Defaults to 250 if unspecified.
<code>description</code>	Brief description for the visualization, appearing in Splunk Web.	No
<code>search_fragment</code>	Brief search portion to indicate how to structure results properly for the visualization. Used in Splunk Web.	No
<code>allow_user_selection</code>	Whether the visualization should be available for users to select in Splunk Web.	No. Defaults to true, meaning the visualization is available.
<code>disabled</code>	If set to 1, the visualization is not available anywhere in the Splunk platform. In this case, overrides a true setting for <code>allow_user_selection</code> .	No. Defaults to 0 if unspecified, meaning that the visualization is available.

---

## Export the visualization

By default, a custom visualization is only available within its own app context. Export the visualization to make it available globally, including to the **Search and Reporting** app. To export the custom visualization app, follow these steps.

1. In the `viz_tutorial_app/metadata` folder, find the `default.meta` file.
2. Open the file and add the following content.

```
[visualizations/radial_meter]
export = system
```

Note that the stanza name syntax is `visualizations/<visualization_folder_name>`. In this case, the visualization folder name is `radial_meter`.

---

## Try out the visualization

The essential visualization rendering code is in place. Now you can build the visualization and run a search.

## Rebuild the visualization


Every time you update the source code in `visualization_source.js`, you must rebuild the visualization to see your changes in Splunk Web.

If developer mode is enabled, it is not necessary to restart the Splunk platform to see changes from the rebuilt visualization in Splunk Web.

### Prerequisites

Ensure that the `$SPLUNK_HOME` environment variable is pointing to the Splunk installation folder. Use the command `echo $SPLUNK_HOME` in a terminal window to verify that the Splunk installation folder path prints. If it does not, set it with this export command: `export SPLUNK_HOME=/Applications/Splunk.`

### Steps

1. Build the visualization by running `$ npm run build` from the `/radial_meter` directory. Notice that this generates the built `visualization.js` file in the same directory.
2. From the **Search and Reporting** home page, run this search.  
`index=_internal | stats count`
3. Select the **Visualizations** tab.
4. Select the Visualization Picker at left to review the available visualizations. The Radial Meter visualization appears under **More**. Note that this tutorial does not include adding a visualization icon, so the Radial Meter visualization uses this generic icon: .
5. Select the Radial Meter visualization to see it render the search results.

### Error handling

If you run a different search than the one shown here, results might not be formatted to generate the visualization properly. You might notice that there are no error messages to indicate this problem to a user. The next tutorial steps show you how to add data format error handling to the Radial Meter visualization.

---

## Handle data format errors

To introduce some data format error handling, go back to `visualization_source.js`. The next steps show you how to override the `formatData` method that `visualization_source.js` inherits from `SplunkVisualizationBase`.

### Override `formatData`

`formatData` gets a raw data object from `splunkd` and returns an object formatted for rendering. This object passes to `updateView` as its `data` argument. Add the following code to `visualization_source.js` to specify how `formatData` validates and processes raw data.

Make sure to add the new `formatData` code exactly as shown. The `...` symbols indicate code already added.

```
getInitialDataParams: function() {  
    ...  
},  
  
formatData: function(data, config) {  
    // Check for an empty data object
```



```

    if(data.rows.length < 1){
        return false;
    }
    var datum = SplunkVisualizationUtils.escapeHtml(parseFloat(data.rows[0][0]));

    // Check for invalid data
    if(_.isNaN(datum)){
        throw new SplunkVisualizationBase.VisualizationError(
            'This meter only supports numbers'
        );
    }

    return datum;
},

updateView: function(data, config) {
    ...

}

...

```

---

### ***Best practices for data validation and handling***

The code just added to `formatData` does not do much formatting. However, it demonstrates important practices for data validation and error handling. Review these best practices when creating any custom visualization.

- **Check for empty data**  
Whenever the search results change, `formatData` is called. Sometimes the results data object is empty and this case needs to be handled.
- **Check for invalid data**  
Handle cases in which the visualization search does not generate data in the correct format for rendering. Check for the expected data format before trying to render the visualization. In this example, `formatData` checks for a number.
- **Throw helpful errors**  
When a visualization throws a `SplunkVisualizationBase.VisualizationError`, users see the error in Splunk Web. Errors can provide information about what went wrong and help users to troubleshoot.
- **Sanitize values added to the DOM**  
Any dynamic value that will be added to the DOM should be passed through `escapeHtml( )` for security.

### ***Change rendering on data format errors***

In addition to these updates to `formatData`, change `updateView` so that it does nothing if `formatData` does not return anything to render.

Replace all of the code in `updateView` with the following code.

```

updateView: function(data, config) {

    // Return if no data
    if (!data) {
        return;
    }

    // Assign datum to the data object returned from formatData

```

```

var datum = data;

// Clear the div
this.$el.empty();

// Pick a color for now
var mainColor = 'yellow';

// Set domain max
var maxValue = 100;

// Set height and width
var height = 220;
var width = 220;

// Create a radial scale representing part of a circle
var scale = d3.scale.linear()
    .domain([0, maxValue])
    .range([- Math.PI * .75, Math.PI * .75])
    .clamp(true);

// Create parameterized arc definition
var arc = d3.svg.arc()
    .startAngle(function(d) {
        return scale(0);
    })
    .endAngle(function(d) {
        return scale(d);
    })
    .innerRadius(70)
    .outerRadius(85);

// SVG setup
var svg = d3.select(this.el).append('svg')
    .attr('width', width)
    .attr('height', height)
    .style('background', 'white')
    .append('g')
    .attr('transform', 'translate(' + width / 2 + ', ' + height / 2 + ')');

// Background arc
svg.append('path')
    .datum(maxValue)
    .attr('d', arc)
    .style('fill', 'lightgray');

// Fill arc
svg.append('path')
    .datum(datum)
    .attr('d', arc)
    .style('fill', mainColor);

// Text
svg.append('text')
    .datum(datum)
    .attr('class', 'meter-center-text')
    .style('text-anchor', 'middle')
    .style('fill', mainColor)
    .text(function(d) {
        return parseFloat(d);
    })
    .attr('transform', 'translate(' + 0 + ', ' + 20 + ')');

```

```
}  
updateView now checks for null data and does not render in this case.
```

After these updates, if a search does not return numbers a useful error message appears.

Run `$npm run build` to rebuild the `visualization.js` file if you want to try out the visualization at this point.

---

## Add user-configurable properties

Custom visualizations can include user-configurable properties and an interface for users to specify settings. This part of the tutorial shows you how to declare two configurable properties and handle property settings in `visualization_source.js`. These steps set up the visualization to work with a user interface.

### *Property namespacing*

Property namespacing follows this syntax for configuration file declarations.

```
display.visualizations.custom.<app_name>.<visualization_ name>.<property_name>
```

When building the `formatter.html` user interface, developers can use a shortened namespace syntax for property references. This syntax option is shown later in this tutorial.

---

### *Property naming*

In configuration files, property names refer to the specific part of the visualization that they affect. For example, use this name for the property that determines the main color of the radial meter.

```
display.visualizations.custom.viz_tutorial_app.radial_meter.mainColor
```

Use this name for the property determining the meter maximum count value.

```
display.visualizations.custom.viz_tutorial_app.radial_meter.maxValue
```

---

### *Declare property information*

Once you determine names for the configurable properties, the next step is to declare the property names, types, and default values.

The radial meter app needs properties for its dial color and maximum value. Add the following property names and types to `viz_tutorial_app/README/savedsearches.conf.spec`.

```
display.visualizations.custom.viz_tutorial_app.radial_meter.mainColor = <string>  
display.visualizations.custom.viz_tutorial_app.radial_meter.maxValue = <float>
```

Next, specify property default values by adding the following content to `viz_tutorial_app/default/savedsearches.conf`.

```
[default]  
display.visualizations.custom.viz_tutorial_app.radial_meter.mainColor = #f7bc38  
display.visualizations.custom.viz_tutorial_app.radial_meter.maxValue = 100
```

Now you can add code to `visualization_source.js` for handling property configurations.

---

### ***Handle property settings***

This part of the tutorial shows you how to use the following two properties to get user settings for the foreground color and the maximum radial meter value.

- `display.visualizations.custom.viz_tutorial_app.radial_meter.mainColor`
- `display.visualizations.custom.viz_tutorial_app.radial_meter.maxValue`

Start by changing `updateView` to check for the properties and use default values in case the properties are not set.

Replace these lines in `updateView`

```
// Pick a color for now
var mainColor = 'yellow';

// Set domain max
var maxValue = 100;
```

with this code.

```
updateView: function(data, config) {
    ...

    // Get color config or use a default yellow shade
    var mainColor = config[this.getPropertyNamespaceInfo().propertyNamespace + 'mainColor'] ||
    '#f7bc38';

    // Set meter max value or use a default
    var maxValue = parseFloat(config[this.getPropertyNamespaceInfo().propertyNamespace + 'maxValue']) ||
    100;
    ...
}
```

After these updates, run `$ npm run build` from the `/radial_meter` directory to rebuild the visualization. At this point, the visualization is ready for a configuration user interface.

---

### **Implement a format menu**

The code now checks for two visualization properties and handles user settings. The next step is to provide users with a user interface for setting these properties. The following steps show you how to define a format menu in HTML and using Splunk Web components. The menu appears in the **Search** page and in dashboard menus.

#### ***Define the format menu***

The format menu defined here has two sections. The first one lets users specify a maximum meter value. The second section uses Splunk Web components to define a color picker for the meter foreground.

## Steps

1. Locate the `viz_tutorial_app/appserver/static/visualizations/radial_meter/formatter.html` file.

2. Open the file and add the following content to it.

```
<form class="splunk-formatter-section" section-label="Max value">
  <splunk-control-group label="Maximum dial value">
    <splunk-text-input name="{VIZ_NAMESPACE}}.maxValue" value="100">
    </splunk-text-input>
  </splunk-control-group>
</form>
<form class="splunk-formatter-section" section-label="Dial color">
  <splunk-control-group label="Color">
    <splunk-color-picker name="{VIZ_NAMESPACE}}.mainColor" value="#f7bc38">
    </splunk-color-picker>
  </splunk-control-group>
</form>
```

---

### Working with `formatter.html`

The code just added shows some important aspects of implementing a format menu.

- Each menu section needs its own form.
- For each section to render separately, make sure to assign each section the `splunk-formatter-section` class.
- Each input element should be named according to the property that it impacts.
- The `splunk-color-picker` component defaults to using the `splunkCategorical` color palette if no `palette type` attribute is specified.
- When a user changes a setting in the format menu, the `config` dictionary gets the updated property value. `updateView` is called using the new values in `config`.

Additionally, the code just added demonstrates using the shortened property namespacing syntax available in `formatter.html`.

### Formatter property namespacing

In `formatter.html`, you can refer to visualization properties using the following shortened namespace syntax.

```
{{VIZ_NAMESPACE}}.<property_name>
```

This shortened syntax is equivalent to the fully qualified name used to declare the property in `savedsearches.conf`.

```
display.visualizations.custom.<app_name>.<viz_name>.<property_name>
```

The following procedure for defining the format menu includes the shortened syntax.

**Note:** The shortened property name syntax is available only in the `formatter.html` file. Use the fully qualified property name in any other visualization app files.

For more information on implementing and configuring the format menu interface, see the [Formatter API reference](#).

---

## Conclusion

You now have a working example custom visualization. You can use the design patterns and best practices in this tutorial to build any custom visualization.

To learn more about creating custom visualizations, see the following topics.

- [Custom visualization API reference](#)
- [Design guidelines](#)

## Custom visualization API reference

The custom visualization API has been updated for Splunk software version 6.5 and is now fully supported. If you are building a new custom visualization app, use the latest version of the API.

Developers whose apps use the experimental API offered with software version 6.4 are encouraged to update their apps. See [API updates and migration advice](#) for more information.

Use this reference to review custom visualization requirements, components, and configuration information.

## App directory structure

To add a custom visualization to an app, put the required visualization package and configuration files into the app directory. The following layout shows required custom visualization components within an app directory. It includes Webpack configuration files.

```
appname
  appserver
    static
      visualizations
        <visualization_name>
          src
            visualization_source.js
            webpack.config.js
            visualization.js
            visualization.css
            formatter.html
            package.json
            preview.png
      default
        visualizations.conf
        savedsearches.conf
      metadata
        default.meta
      README
        savedsearches.conf.spec
```

---

## App packaging

### Webpack

Use Webpack to package custom visualization apps to run on the Splunk platform.

Webpack is a resource packaging tool that can build multiple app dependencies into a single package. This functionality provides isolation, encapsulation, and improved performance.

To learn more about Webpack, see <https://webpack.github.io>.

### Template Webpack configuration

You can use or extend the custom visualization app template to build any custom visualization app. This template includes a `visualization_source.js` file and an initial Webpack configuration that works with `npm` (node.js package manager).

### Webpack configuration details

The template Webpack configuration is located in the stand-in visualization package. You can find it in the following folder.  
`$TEMPLATE_APP_ROOT/appserver/static/visualizations/standin/`.

The configuration contains the following two files.

File	Description	Details
<code>package.json</code>	npm package file.	<ul style="list-style-type: none"><li>• Declares the visualization package dependencies and useful shortcuts for running build tasks.</li></ul>
<code>webpack.config.js</code>	Webpack configuration file.	<ul style="list-style-type: none"><li>• Defines the build entry point and the output location</li><li>• Can be extended to load various resources into a visualization package.</li><li>• By default, <code>webpack.config.js</code> defines the source entry point of the package as <code>src/visualization_source.js</code> and the output file as <code>visualization.js</code>.</li><li>• When Webpack runs in the same directory as this configuration file, it builds the JavaScript and its dependencies together and produces a built <code>visualization.js</code> file.</li></ul>

### Webpack a custom visualization

Use the Webpack configuration in the custom visualization template to build custom visualizations. After Webpack builds the custom visualization `visualization.js` file, the template app package can be run as an app on the Splunk platform.

### Prerequisites

- Make sure that `npm` is installed. To install `npm`, see [www.npmjs.com](http://www.npmjs.com).
- Download the custom visualization template and review the Webpack configuration files.

### Steps

1. In the custom visualization template, rename the `/standin` visualization directory for the custom visualization you are building.

2. Add your custom visualization code to `visualization_source.js` file.
3. Update `package.json`
  - ◆ Change the visualization name to match the custom visualization you are building.
  - ◆ Add any custom visualization dependencies to the dependencies list.
  - ◆ Change the description and any other details relevant to your custom visualization.
4. From your custom visualization directory run `$ npm install`.
5. From the same directory run `$ npm run build`.

These steps should produce the following built visualization file.

```
$TEMPLATE_APP_ROOT/appserver/static/visualizations/<visualization_name>/visualization.js
```

---

## Logic

Use these components to build custom visualization logic.

### *visualization.js*

By default, this file is built from the `visualization_source.js` source code using Webpack.

### *visualization\_source.js*

#### Description

This visualization source code file contains the central logic for capturing and rendering data in the visualization. It is an AMD (Asynchronous Module Definition) module that returns an object extending `SplunkVisualizationBase`.

#### Requirements

Component	Requirement details	For more information see
<b>SplunkVisualizationBase</b>	<p><code>visualization_source.js</code> must override the following functions and options in this class.</p> <ul style="list-style-type: none"> <li>• <code>getInitialDataParams</code>. Indicates how the visualization framework fetches data for the visualization.</li> <li>• <code>updateView</code>. Function called to render the visualization.</li> </ul>	<a href="#">SplunkVisualizationBase</a>
<b>SplunkVisualizationUtils</b>	<p><b>Security requirement</b></p> <p><code>visualization_source.js</code> must include this utility library for safely incorporating dynamic content in visualizations.</p>	<a href="#">Security utilities</a>

---

### *SplunkVisualizationBase*

The `SplunkVisualizationBase` class offers an API for access and communication with the Splunk platform.

#### Requirements

`visualization_source.js` extends this class. It must override the following methods and options.



updateView

This function is called whenever search results are updated or the visualization format changes. It handles visualization rendering using the following two parameters.

- **data.** An object containing search result data.
- **config.** An object containing visualization format information.

getInitialDataParams

This function is required for data to be returned from the search. It specifies the data output format for search results. You can also use this function to specify the maximum number of results. Use any of the following data output options.

Mode	Example
<b>Row-major</b>  List field values as they appear in each row of the raw JSON results object array.  Returns a field name array and a row array. Each row array index contains an array representing all field values for one result.  <code>getInitialDataParams</code> option name  <code>SplunkVisualizationBase.ROW_MAJOR_OUTPUT_MODE</code>	<pre>{   fields: [     { name: 'store_id' },     { name: 'qty' },     { name: 'product' }   ],   rows: [     ['west', 400, 'shirt'],     ['east', 625, 'mug'],     ['north', 812, 'hat']   ] }</pre>
<b>Column-major</b>  List field values as they appear in each column of the raw JSON results object array.  Returns a field name array and a column array. Each column array index contains an array representing all results values for one field.  <code>getInitialDataParams</code> option name  <code>SplunkVisualizationBase.COLUMN_MAJOR_OUTPUT_MODE</code>	<pre>{   fields: [     { name: 'store_id' },     { name: 'qty' },     { name: 'product' }   ],   columns: [     ['west', 'east', 'north'],     [400, 625, 812],     ['shirt', 'mug', 'hat']   ] }</pre>
<b>Raw format</b>  Raw JSON object.  Returns a field name array and a results array in which each index contains a result object showing all fields and field values as key value pairs.  <code>getInitialDataParams</code> option name	<pre>{   fields: [     { name: 'store_id' },     { name: 'qty' },     { name: 'product' }   ],   results: [     { store_id: 'west', qty: 400, product: 'shirt'},     { store_id: 'east', qty: 625, product: 'mug'},   ] }</pre>

Mode	Example
<code>SplunkVisualizationBase.RAW_OUTPUT_MODE</code>	<pre>       { store_id: 'north', qty: 812, product: 'hat'},     ]   } </pre>

## Interface methods

The following methods are available in `SplunkVisualizationBase`. See the interface code to review parameters for each method.

Method	Description	Override required?
<code>initialize</code>	<ul style="list-style-type: none"> <li>• Override to define initial data parameters that the framework should use to fetch data for the visualization.</li> <li>• The code in this method can assume that the visualization root DOM (Document Object Model) element is available as <code>this.el</code>.</li> </ul>	Optional
<code>getInitialDataParams</code>	<ul style="list-style-type: none"> <li>• Override to define the initial data parameters used to fetch data for the visualization.</li> </ul>	<b>Required</b>
<code>onConfigChange</code>	<ul style="list-style-type: none"> <li>• Override to implement custom handling of the <code>config</code> attribute changes.</li> <li>• The default behavior is to mark <code>formatData</code> invalid.</li> </ul>	Optional
<code>formatData</code>	<ul style="list-style-type: none"> <li>• Override to implement custom data processing logic. The return value passes to <code>updateView</code>.</li> <li>• The <code>data</code> return value includes a <code>data.meta.done</code> Boolean flag indicating that the search job is complete.</li> <li>• Do not call this method directly in <code>visualization_source.js</code>. Call <code>invalidateFormatData</code> instead to indicate that <code>formatData</code> needs to run again.</li> <li>• The framework batches all invalidation methods to update the visualization efficiently.</li> </ul>	Optional
<code>setupView</code>	<ul style="list-style-type: none"> <li>• Override to implement the initial view setup logic.</li> <li>• This method is called immediately before the first call to <code>updateView</code>.</li> <li>• Do not call this method directly in <code>visualization_source.js</code>.</li> </ul>	Optional
<code>updateView</code>	<ul style="list-style-type: none"> <li>• Override to implement visualization rendering logic.</li> <li>• Do not call this method directly in <code>visualization_source.js</code>. Call <code>invalidateUpdateView</code> instead to indicate that <code>updateView</code> needs to run again.</li> <li>• The framework batches all invalidation methods to update the visualization efficiently.</li> </ul>	<b>Required</b>
<code>reflow</code>	<ul style="list-style-type: none"> <li>• Override to implement visualization resizing logic.</li> <li>• This method is called whenever the container dimensions change. Measure <code>this.el</code> to determine current dimensions.</li> <li>• Do not call this method directly in <code>visualization_source.js</code>. Call <code>invalidateReflow</code> instead to indicate that <code>reflow</code> needs to run again.</li> </ul>	Optional

Method	Description	Override required?
	<ul style="list-style-type: none"> <li>The framework batches all invalidation methods to update the visualization efficiently.</li> </ul>	
<code>remove</code>	<ul style="list-style-type: none"> <li>Override to perform necessary disassembly logic.</li> </ul>	Optional
<code>updateDataParams</code>	<ul style="list-style-type: none"> <li>Call this method to update the data parameters to use when fetching data. The framework fetches an updated data set.</li> </ul>	Do not override. Treat this method as final.
<code>drilldown</code>	<ul style="list-style-type: none"> <li>Call this method when a drilldown interaction happens.</li> </ul>	Do not override. Treat this method as final.
<code>invalidateFormatData</code>	<ul style="list-style-type: none"> <li>Call this method to indicate that <code>formatData</code> needs to run again.</li> <li>The framework batches all invalidation methods to update the visualization efficiently.</li> </ul>	Do not override. Treat this method as final.
<code>invalidateUpdateView</code>	<ul style="list-style-type: none"> <li>Call this method to indicate that <code>updateView</code> needs to run again.</li> <li>The framework batches all invalidation methods to update the visualization efficiently.</li> </ul>	Do not override. Treat this method as final.
<code>invalidateReflow</code>	<ul style="list-style-type: none"> <li>Call this method to indicate that <code>reflow</code> needs to run again.</li> <li>The framework batches all invalidation methods to update the visualization efficiently.</li> </ul>	Do not override. Treat this method as final.
<code>getCurrentData</code>	<ul style="list-style-type: none"> <li>Call this method to get the current data returned by <code>formatData</code>.</li> </ul>	Do not override. Treat this method as final.
<code>getCurrentConfig</code>	<ul style="list-style-type: none"> <li>Call this method to get current configuration attributes.</li> </ul>	Do not override. Treat this method as final.
<code>getPropertyNamespaceInfo</code>	<ul style="list-style-type: none"> <li>Call this method to get visualization namespace information.</li> </ul>	Do not override. Treat this method as final.
<code>setCurrentData</code>	<ul style="list-style-type: none"> <li>Reserved method name.</li> </ul>	Do not call or override.
<code>setCurrentConfig</code>	<ul style="list-style-type: none"> <li>Reserved method name.</li> </ul>	Do not call or override.

**SplunkVisualizationBase** public interface

## SplunkVisualizationBase

```
define([
    'underscore',
    'backbone'
],
function(
    _
    Backbone
) {

    var VisualizationError = function(message) {
        this.name = 'SplunkVisualizationError';
        this.message = message || '';
        Error.apply(this, arguments);
    };
});
```

```

VisualizationError.prototype = new Error();

var SplunkVisualizationBase = function(el, appName, vizName) {
  this.el = el;
  this._config = null;
  this._data = null;
  this._appName = appName;
  this._vizName = vizName;
  this.initialize();
};

_.extend(SplunkVisualizationBase.prototype, Backbone.Events, {

  /**
   * Override to perform constructor logic.
   *
   * Code in initialize can assume that the visualization has been assigned
   * a root DOM element, available as `this.el`.
   */
  initialize: function() {},

  /**
   * Override to define initial data parameters that the framework should use to
   * fetch data for the visualization.
   *
   * Allowed data parameters:
   *
   * outputMode (required) the data format that the visualization expects, one of
   * - SplunkVisualizationBase.COLUMN_MAJOR_OUTPUT_MODE
   *   {
   *     fields: [
   *       { name: 'x' },
   *       { name: 'y' },
   *       { name: 'z' }
   *     ],
   *     columns: [
   *       ['a', 'b', 'c'],
   *       [4, 5, 6],
   *       [70, 80, 90]
   *     ]
   *   }
   * - SplunkVisualizationBase.ROW_MAJOR_OUTPUT_MODE
   *   {
   *     fields: [
   *       { name: 'x' },
   *       { name: 'y' },
   *       { name: 'z' }
   *     ],
   *     rows: [
   *       ['a', 4, 70],
   *       ['b', 5, 80],
   *       ['c', 6, 90]
   *     ]
   *   }
   * - SplunkVisualizationBase.RAW_OUTPUT_MODE
   *   {
   *     fields: [
   *       { name: 'x' },
   *       { name: 'y' },
   *       { name: 'z' }
   *     ],
   *     results: [

```

```

*           { x: 'a', y: 4, z: 70 },
*           { x: 'b', y: 5, z: 80 },
*           { x: 'c', y: 6, z: 90 }
*       ]
*   }
*
* count (optional) how many rows of results to request, default is 1000
*
* offset (optional) the index of the first requested result row, default is 0
*
* sortKey (optional) the field name to sort the results by
*
* sortDirection (optional) the direction of the sort, one of:
* - SplunkVisualizationBase.SORT_ASCENDING
* - SplunkVisualizationBase.SORT_DESCENDING (default)
*
* search (optional) a post-processing search to apply to generate the results
*
* @param {Object} config The initial config attributes
* @returns {Object}
*
*/
getInitialDataParams: function(config) {
    return {};
},

/**
* Override to implement custom handling of config attribute changes.
*
* Default behavior is to mark the formatData routine invalid.
*
* @param {Object} configChanges The changed config attributes, an object with
*     changed keys mapping to their new values
* @param {Object} previousConfig The previous config attributes
*/
onConfigChange: function(configChanges, previousConfig) {
    this.invalidateFormatData();
},

/**
* Override to implement custom data processing logic.
*
* The return value of this method will be passed to the updateView routine.
* This method should not be called directly by visualization code, call
* invalidateFormatData instead to notify the framework that the formatData
* routine needs to be run again.
*
* @param {Object} rawData The data in its raw form
* @param {Object} config The current config attributes
* @returns {*}
*/
formatData: function(rawData, config) {
    return rawData;
},

/**
* Override to implement one-time view setup logic.
*
* This method will be called immediately before the first call to the
* updateView routine.
* This method should not be called directly by visualization code.
*/

```

```

setupView: function() {},

/**
 * Override to implement visualization rendering logic.
 *
 * This method should not be called directly by visualization code, call
 * invalidateUpdateView instead to notify the framework that the updateView
 * routine needs to be run again.
 *
 * @param {*} data The formatted data, as returned by the formatData routine
 * @param {Object} config The current config attributes
 * @param {Function} async A function that notifies the framework that the
 *   visualization will update asynchronously.
 *   If all updates are occurring synchronously within updateView,
 *   the `async` parameter can be ignored.
 *   If any updates are asynchronous (e.g. animations), call async() and
 *   use the return value as a callback to signal that the update has completed:
 *
 *   updateView: function(data, config, async) {
 *     var done = async();
 *     this.performAsyncUpdates({
 *       onComplete: done
 *     });
 *   }
 */
updateView: function(data, config, async) {},

/**
 * Override to implement visualization resizing logic.
 *
 * This method will be called whenever the container dimensions change.
 * The current container dimensions can be obtained by measuring `this.el`.
 * This method should not be called directly by visualization code, call
 * invalidateReflow instead to notify the framework that the reflow
 * routine needs to be run again.
 */
reflow: function() {},

/**
 * Override to perform all necessary teardown logic.
 */
remove: function() {},

/**
 * Call this method to update the data parameters to be used when fetching data,
 * the framework will fetch an updated data set.
 *
 * This method should be treated as final.
 *
 * @param {Object} newParams New data parameters, to be merged with the existing ones.
 *   See getInitialData above for a description of allowed inputs.
 */
updateDataParams: function(newParams) {
  this.trigger('updateDataParams', newParams);
},

/**
 * Call this method to notify the framework of a drilldown interaction.
 *
 * @param payload {Object} a description of the "intention" of the drilldown interaction.
 */

```

```

* Two different type of drilldown action are supported:
*
* 1) Field-value pair drilldown, where the "intention" is to filter the results by
*    setting one or more field-value pairs as constraints, e.g.
*
*    this.drilldown({
*      action: SplunkVisualizationBase.FIELD_VALUE_DRILLDOWN,
*      data: {
*        fieldOne: valueOne,
*        fieldTwo: valueTwo,
*        ...
*      }
*    });
*
* 2) Geo-spatial drilldown, where the "intention" is to filter the results to a
*    geo-spatial region, e.g.
*
*    this.drilldown({
*      action: SplunkVisualizationBase.GEOSPATIAL_DRILLDOWN,
*      data: {
*        lat: {
*          name: <name of latitude field>
*          value: <value of latitude field>
*        },
*        lon: {
*          name: <name of longitude field>
*          value: <value of longitude field>
*        },
*        bounds: [<south>, <west>, <north>, <east>]
*      }
*    });
*
* Additionally, the "intention" can filter the results to a specific time range.
* The time range can be combined with any of the actions above, e.g.
*
*    this.drilldown({
*      earliest: '1981-08-18T00:00:00.000-07:00',
*      latest: '1981-08-19T00:00:00.000-07:00'
*      // optionally an `action` and `data`
*    });
*
* The `earliest` and `latest` values can be ISO timestamps in the
* format above, or as epoch times.
*
* @param originalEvent {Event} (optional) the original browser event that initiated the
*   interaction, used to support keyboard modifiers.
*
* This method should be treated as final.
*/

drilldown: function(payload, originalEvent) {
  this.trigger('drilldown', payload, originalEvent);
},

/**
* Call this method to notify the framework that the formatData routine needs to run again.
*
* The framework batches calls to this and other invalidation methods so that
* the visualization will be updated efficiently.
*
* This method should be treated as final.
*/

```

```

invalidateFormatData: function() {
    this.trigger('invalidateFormatData');
},

/**
 * Call this method to notify the framework that the updateView routine needs to run again.
 *
 * The framework batches calls to this and other invalidation methods so that
 * the visualization will be updated efficiently.
 *
 * This method should be treated as final.
 */
invalidateUpdateView: function() {
    this.trigger('invalidateUpdateView');
},

/**
 * Call this method to notify the framework that the reflow routine needs to run again.
 *
 * The framework batches calls to this and other invalidation methods so that
 * the visualization will be updated efficiently.
 *
 * This method should be treated as final.
 */
invalidateReflow: function() {
    this.trigger('invalidateReflow');
},

/**
 * Call this method to get the current data, as returned by the formatData routine.
 * Cannot be called in initialize.
 *
 * This method should be treated as final.
 *
 * @returns {*}
 */
getCurrentData: function() {
    return this._data;
},

/**
 * Call this method to get the current config attributes.
 * Cannot be called in initialize.
 *
 * This method should be treated as final.
 *
 * @returns {Object}
 */
getCurrentConfig: function() {
    return this._config;
},

/**
 * Call this method to get info about the viz namespace.
 *
 * This method should be treated as final.
 *
 * @returns {
 *     appName: <string>,
 *     vizName: <string>,
 *     propertyNameSpace: <string>
 * }
 */

```



```

    */
    getPropertyNamespaceInfo: function() {
        return {
            appName: this._appName,
            vizName: this._vizName,
            propertyNamespace: 'display.visualizations.custom.' + this._appName + '.' + this._vizName +
            '.';
        };
    },

    /**
     * Used internally for communication between the framework and the visualization.
     *
     * This method should be treated as final, and should not be called by visualization code.
     */
    setCurrentData: function(data) {
        this._data = data;
    },

    /**
     * Used internally for communication between the framework and the visualization.
     *
     * This method should be treated as final, and should not be called by visualization code.
     */
    setCurrentConfig: function(config) {
        this._config = config;
    }

});

_.extend(SplunkVisualizationBase, {
    extend: Backbone.View.extend,

    COLUMN_MAJOR_OUTPUT_MODE: 'json_cols',
    ROW_MAJOR_OUTPUT_MODE: 'json_rows',
    RAW_OUTPUT_MODE: 'json',

    FIELD_VALUE_DRILLDOWN: 'fieldvalue',
    GEOSPATIAL_DRILLDOWN: 'geoviz',

    SORT_ASCENDING: 'asc',
    SORT_DESCENDING: 'desc',

    // Defines a custom error type to be thrown by sub-class in order to
    // propagate an error message up to the user of the visualization, e.g.
    //
    // if (data.columns.length < 2) {
    //     throw new SplunkVisualizationBase.VisualizationError(
    //         'This visualization requires at least two columns of data.'
    //     );
    // }
    VisualizationError: VisualizationError
});

return SplunkVisualizationBase;
});

```

---

## Drilldown options

Use the provided `drilldown` function in `SplunkVisualizationBase` to implement a time based or field-value drilldown. You can also combine these drilldown types.

### Time based drilldown

A time based drilldown lets users click on the visualization to open a search over the time range that the clicked area represents. Determine the earliest and latest time. Call `this.drilldown` and pass the time parameters to the `drilldown` function.

Here is an example that incorporates a helper function.

```
/**
 * To be called from the visualization's click handler, after computing the
 * correct time range from the target of the click.
 *
 * @param earliestTime - the lower bound of the time range,
 *                       can be an ISO-8601 timestamp or an epoch time in seconds.
 * @param latestTime - the upper bound of the time range,
 *                     can be an ISO-8601 timestamp or an epoch time in seconds.
 * @param browserEvent - the original browser event that caused the drilldown
 *
 * example usage:
 *
 * this.drilldownToTimeRange('1981-08-18T00:00:00.000-07:00', '1981-08-19T00:00:00.000-07:00', e);
 */
drilldownToTimeRange: function(earliestTime, latestTime, browserEvent) {
    this.drilldown({
        earliest: earliestTime,
        latest: latestTime
    }, browserEvent);
}
```

### Field-value drilldown

A field-value drilldown lets users click on the visualization to open a search using one or more category name and value pairs that the clicked area represents. Determine the category names and values. Call `this.drilldown` and pass the name and value parameters to the `drilldown` function.

Here is an example that incorporates a helper function. This example uses one field-value pair but the custom visualizations API supports using multiple pairs.

```
/**
 * To be called from the visualization's click handler, after computing the
 * correct category name and value from the target of the click.
 *
 * @param categoryName - the field name for the category
 * @param categoryFieldValue - the value for the category
 * @param browserEvent - the original browser event that caused the drilldown
 *
 * example usage:
 *
 * this.drilldownToTimeRange('State', 'Oregon', e);
 */
drilldownToCategory: function(categoryName, categoryFieldValue, browserEvent) {
    var data = {};
```

```

data[categoryName] = categoryFieldValue;

this.drilldown({
  action: SplunkVisualizationBase.FIELD_VALUE_DRILLDOWN,
  data: data
}, browserEvent);
}

```

## Time based and field-value drilldown example

You can combine the two drilldown options as shown in this example.

```

drilldownToTimeRangeAndCategory: function(earliestTime, latestTime, categoryName, categoryValue,
browserEvent) {
  var data = {};
  data[categoryName] = categoryValue;

  this.drilldown({
    action: SplunkVisualizationBase.FIELD_VALUE_DRILLDOWN,
    data: data,
    earliest: earliestTime,
    latest: latestTime
  }, browserEvent);
}

```

## Utility functions

The `SplunkVisualizationUtils` class provides utility and security functions for custom visualizations. In addition to the following utility functions, see [Security utilities](#) for information on required security functions.

### Utility

The following utility functions are available in `SplunkVisualizationUtils`.

Function	Description	Arguments	Returns
<code>getColorPalette</code>	Get a predefined color palette for categorical or semantic colors. See <a href="#">Design guidelines</a> for more details on color palettes.	<p><code>name</code> (String), required. Use of the following palette names:</p> <ul style="list-style-type: none"> <li><code>splunkCategorical</code> (default)</li> <li><code>splunkSemantic</code></li> </ul> <p>You can also use one of the following themes: <code>theme</code> (String), optional:</p> <ul style="list-style-type: none"> <li><code>light</code> (default)</li> <li><code>dark</code></li> </ul>	Array of color strings in the specified palette. If no argument is provided, the function returns the <code>splunkCategorical</code> color palette. Adding a theme option, for example, <code>dark</code> , will return the semantic and categorical color palettes available for dark theme.
<code>getCurrentTheme</code>	Use this function to return the current active theme.		Returns <code>dark</code> or <code>light</code> .
<code>parseTimestamp</code>			

Function	Description	Arguments	Returns
	Use this function whenever a date is initialized with a timestamp. The function must be used to show the correct server date and time to a user.	<code>timestamp</code> (ISO string object)	Timezone-corrected JavaScript date object. If a non-ISO string is provided, the function returns a date. Timezone correction is not possible in this case.
<code>normalizeBoolean</code>	<p>Use this function for strict normalization of a String to a Boolean.</p> <p>The following values are handled as <code>true</code>.</p> <p>String (case-insensitive):  <code>"true"</code>, <code>"on"</code>, <code>"yes"</code>, <code>"1"</code></p> <p>Integer: <code>1</code></p> <p>Boolean: <code>true</code></p> <p>The following values are treated as <code>false</code>.</p> <p>String (case-insensitive):  <code>"false"</code>, <code>"off"</code>, <code>"no"</code>, <code>"0"</code></p> <p>Integer: <code>0</code></p> <p>Boolean: <code>false</code></p>	<p><code>value</code> (String, Integer, or Boolean). Value to be normalized.</p> <p><code>options</code> (Object). Default value to return if the expression is not valid.</p>	When a valid boolean expression is submitted, returns the normalized result. If the expression is not valid, returns the <code>options</code> value. If <code>options</code> is not specified, returns <code>false</code> .

## Security utilities

Developers are required to sanitize dynamic content for custom visualization app certification.

Splunk platform searches can have unconstrained data outputs. This means that any data that you present in any context might be malicious. The `SplunkVisualizationUtils` library addresses the following common risks.

- XSS (Cross-site scripting) injection into the DOM (Document Object Model)
- XSS injection using unsafe URL schemes

## Requirements

Make sure that `visualization_source.js` meets the following security requirements.

Requirement	Utility function to use	What to do
Prevent XSS injection.	<code>escapeHtml(inputString)</code>	<p>Before adding any strings to the DOM, pass them through this <code>SplunkVisualizationUtils</code> function.</p> <p>If you are using a framework that handles HTML escaping automatically, you do not have to use this function in addition.</p>
Strip dynamic content from unsafe URL schemes.	<code>makeSafeUrl(inputUrl)</code>	This <code>SplunkVisualizationUtils</code> function is required for app certification if the custom visualization displays links with dynamic

Requirement	Utility function to use	What to do
		URLs.

### Examples

The following examples show you how to use security utility functions in `visualization_source.js`.

#### escapeHtml usage example

```
define([
    'api/SplunkVisualizationBase',
    'api/SplunkVisualizationUtils'
],
function(
    SplunkVisualizationBase,
    SplunkVisualizationUtils
) {
    var SampleViz = SplunkVisualizationBase.extend({
        getInitialDataParams: function() {
            return { outputMode: SplunkVisualizationBase.COLUMN_MAJOR_OUTPUT_MODE };
        },
        updateView: function(data, config) {
            // Both the title and the data point are dynamic values that could potentially
            // contain malicious content (e.g. "<script>alert(1)</script>").
            // The escapeHTML function will encode them so that they can safely
            // be inserted into the DOM.
            var title = config['display.visualizations.custom.sampleApp.sampleViz.title'];
            this.el.innerHTML = '<h1>' + SplunkVisualizationUtils.escapeHtml(title) + '</h1>';
            var dataPoint = data.columns[0][0];
            this.el.innerHTML += '<p>' + SplunkVisualizationUtils.escapeHtml(dataPoint) + '</p>';
        }
    });

    return SampleViz;
});
```

#### makeSafeUrl usage example

```
define([
    'api/SplunkVisualizationBase',
    'api/SplunkVisualizationUtils'
],
function(
    SplunkVisualizationBase,
    SplunkVisualizationUtils
) {
    var SampleViz = SplunkVisualizationBase.extend({
        getInitialDataParams: function() {
            return { outputMode: SplunkVisualizationBase.COLUMN_MAJOR_OUTPUT_MODE };
        },
        updateView: function(data, config) {
            var dataPoint = data.columns[0][0];
            this.el.innerHTML = '<p>' + SplunkVisualizationUtils.escapeHtml(dataPoint) + '</p>';
            var seeMoreUrl = config['display.visualizations.custom.sampleApp.sampleViz.seeMoreUrl'];
            var seeMoreLink = document.createElement('a');
            seeMoreLink.innerHTML = 'See More';
            // The "see more" URL is a dynamic value that could potentially contain
```

```

    // a malicious URL (e.g. "javascript:alert(1)").
    // The makeSafeUrl function will strip out any un-safe URL schemes.
    seeMoreLink.setAttribute('href', SplunkVisualizationUtils.makeSafeUrl(seeMoreUrl));
    this.el.appendChild(seeMoreLink);
  }
});

return SampleViz;
});

```

## User interface

Use these components to build a custom visualization user interface.

### ***visualization.css***

#### **Description**

This file contains CSS rules relevant to the visualization.

#### **Guidelines**

- The file should contain standard CSS.
- The Splunk platform picks up `visualization.css` automatically. The CSS file does not need to be pulled in by the visualization.
- Use `class` attributes instead of `id` attributes to define CSS rules. Dashboards can contain multiple instances of the same visualization.
- Use namespacing to avoid CSS class name conflict. See the following CSS rule namespacing requirements.

#### **CSS class namespacing requirements**

Dashboards can contain multiple visualizations. Scope CSS rules so that they are applied correctly. Use these best practices for constraining CSS rules.

Best practice	What to do
If possible, scope all styles by adding a class name unique to the root DOM element.	Include the app and visualization context in the class name as shown in this example. <pre>.splunk-custom-horizon-chart .axis {    // axis styles here  }</pre>
Another option is to create a class name unique to the app and visualization.	For example, use <pre>.splunk-horizon-chart-axis</pre> instead of <code>.axis</code> .

#### **Example visualization.css**

This CSS file contains text rules using prefixed classes.

```

.custom-radial-meter-chart .center-text {
  font-size: 45px;
}

```

```

font-weight: 200;
font-family: "Helvetica Neue", Helvetica, sans-serif;
}
.custom-radial-meter-chart .under-text {
font-size: 20px;
font-weight: 100;
font-family: "Helvetica Neue", Helvetica, sans-serif;
}

```

---

## ***formatter.html***

### **Description**

Contains HTML to render in the visualization format menu. Input elements in the HTML can specify properties to edit. These changes are passed to the visualization.

### **Guidelines**

- The `formatter.html` file can contain any of the following components.
  - ◆ Multiple forms to render multiple tabs.
  - ◆ HTML only
  - ◆ Splunk platform web components.
- Name input elements according to the namespaced property that they affect. When inputs update the corresponding namespaced property in the page, the changes pass to the visualization.
- Use the following case-sensitive namespace element and syntax to specify a property.
 

```
{{VIZ_NAMESPACE}}.<property name>
```

The `{{VIZ_NAMESPACE}}` element is replaced with the namespaced portion of a property name. When combined with the property name, it works equivalently to a fully-qualified namespace.

The fully-qualified namespace syntax previously used for properties is also still supported.

```
display.visualizations.custom.<app name>.<viz name>.<property name>
```

For complete details, see the [Formatter API reference](#).

---

## ***preview.png***

### **Description**

PNG image file used in the **Visualization Picker** user interface. Users see the image and custom visualization name when choosing a visualization.

### **Guidelines**

- For best results the image size should be 116px wide by 76px high.
- Only PNG formatted images are accepted.
- This file is not used as the app icon. To add an icon, see the following instructions.

## **Adding an app icon and screenshot image**

1. Save a 36px by 36px PNG icon image as `appIcon.png`. Follow the pixel dimensions and file name case and spelling exactly.
2. (Optional). To add a screenshot, save a 623px by 350px PNG image as `screenshot.png`. Follow the pixel dimensions and file name case and spelling exactly.
3. Place the `appIcon.png` and `screenshot.png` files in the `<app_name>/appserver/static` directory.

---

## Configuration and access control

Use these components to define custom visualization configurations and access control.

### *visualizations.conf*

#### Description

To make custom visualizations available across a Splunk deployment, declare them in `visualizations.conf`.

#### Requirements

- Create a stanza for each visualization in `visualizations.conf`. Add the settings for loading the visualization and making it available in Splunk Web.
- Every visualization in an app must have a stanza in the `visualizations.conf` file. **The stanza name and the `label` attribute are required** but there are other optional settings. See the following settings list.

---

#### Available settings

Setting	Description	Required?
<code>label</code>	Public label used throughout Splunk Web to refer to the visualization.	Yes
<code>default_height</code>	Default visualization height.	No. Defaults to 250 if unspecified.
<code>description</code>	Brief description for the visualization, appearing in Splunk Web.	No
<code>search_fragment</code>	Brief search portion to indicate how to structure results for the visualization. Used in Splunk Web.	No
<code>allow_user_selection</code>	Whether the visualization should be available for users to select in Splunk Web.	No. Defaults to <code>true</code> , meaning that the visualization is available. A <code>true</code> setting can be overridden if <code>disabled</code> is set to 1.
<code>disabled</code>	If set to 1, the visualization is not available anywhere in the Splunk platform. In this case, the <code>disabled</code> setting overrides a <code>true</code> setting for <code>allow_user_selection</code> .	No. Defaults to 0 if unspecified, meaning that the visualization is available.
<code>supports_trellis</code>	Indicates whether trellis layout is available for this visualization.	No. Defaults to <code>false</code>
<code>supports_drilldown</code>	Indicates whether drilldown can be configured for this visualization in the drilldown editor user interface.	No. Defaults to <code>false</code>

#### Example `visualizations.conf`

This example defines two different visualizations in two stanzas. The first stanza includes all available settings. The second stanza includes only the required settings.



```
[first_example_visualization]
label = First example
description = Use this visualization for making example charts.
default_height = 500
search_fragment = | stats count by a, b, c, d
```

```
[second_example_visualization]
label = Second example
```

---

## ***savedsearches.conf***

### **Description**

Use `savedsearches.conf` to indicate visualization property default values.

- Note: Set property defaults and handle user-configured property values in `visualization_source.js`.

### **Requirements**

- Specify properties by their complete namespace. See the example below for more details.

### **Example `savedsearched.conf`**

This example sets default values for two properties.

```
display.visualizations.custom.viz_sample_app.sample_viz.numericProperty = 100
display.visualizations.custom.viz_sample_app.sample_viz.stringProperty = stringDefault
```

---

## ***default.meta***

### **Description**

Use this file to export visualizations to the system. For additional details, see the `default.meta` configuration spec file in the *Admin Manual*.

### ***savedsearches.conf.spec***

### **Description**

Use this file to declare visualization properties.

- Note: Set property defaults and handle user-configured property values in `visualization_source.js`.

### **Requirements**

- Declare all properties in this file to ensure proper handling. Properties not declared in `savedsearches.conf.spec` are treated as invalid and prompt warnings at startup if they are used in reports.

### **Example `savedsearched.conf.spec`**

This example declares two properties.

```
display.visualizations.custom.viz_sample_app.sample_viz.numericProperty = <float>
display.visualizations.custom.viz_sample_app.sample_viz.stringProperty = <string>
```

---

## Documentation

Add documentation for each custom visualization in an app.

Add a `README.md` file to the `<visualization_name>` folder. Include details for an admin and end user audience. You can follow this template to cover necessary information.

### Documentation template

Documentation area	What to include
Visualization introduction	<ul style="list-style-type: none"><li>• Visualization title</li><li>• Overview of how the visualization works to represent data</li><li>• Details about what kind of data works best for the visualization</li><li>• Brief use case examples</li></ul>
Search and data formatting	<ul style="list-style-type: none"><li>• How to write queries that generate results in the proper data structure or format for this visualization</li><li>• Example queries</li></ul>
Visualization components	<ul style="list-style-type: none"><li>• How different components add meaning or information to the visualization. For example, describe sparklines and trend indicators in a single value visualization.</li></ul>
Customization options	<ul style="list-style-type: none"><li>• How to use the Format menu to customize and configure the visualization</li></ul>
Simple XML	<ul style="list-style-type: none"><li>• List and define any visualization-specific Simple XML options</li><li>• Document full option names, data type, and any rules for accepted values</li><li>• Example configuration to demonstrate correct usage</li></ul>
Drilldown	<ul style="list-style-type: none"><li>• Details of using drilldown with the visualization</li></ul>
Extensions	<ul style="list-style-type: none"><li>• Details of extending or customizing the visualization using available frameworks.</li></ul>
Permissions or other administrative information	<ul style="list-style-type: none"><li>• Details for admins about managing the app for end users</li></ul>
Support contact	<ul style="list-style-type: none"><li>• Developer email or other contact details to use for questions and feedback</li></ul>
Software credits	<ul style="list-style-type: none"><li>• Include citation details for third party software libraries or other tools used in the app. For example: <a href="https://d3js.org/">https://d3js.org/</a></li></ul>

---

## Additional API interactions

You can use custom visualizations in SimpleXML dashboards and SplunkJS.

## Simple XML

You can include custom visualizations in Simple XML dashboards. For more information see [Use custom visualizations in Simple XML](#).

## SplunkJS

Custom visualization components registered with the system are accessible from SplunkJS. For more information see [Use custom visualizations in SplunkJS](#).

---

## Additional resources

For design and data handling best practices, see the following topics.

- [Design guidelines](#)
- [Data handling guidelines](#)

## Formatter API reference

The custom visualization API has been updated for Splunk software version 6.5 and is now fully supported. If you are building a new custom visualization app, use the latest version of the API.

Developers whose apps use the experimental API offered with software version 6.4 are encouraged to update their apps. See [API updates and migration advice](#) for more information.

Use the Formatter API to build a custom visualization format menu. The Formatter API supports user experience consistency with the Splunk Web visualization Format menu.

## Custom HTML element overview

The Splunk platform supports a set of custom HTML elements that manage the behavior and rendering of user interface controls. Here is an overview of the available elements.

Custom HTML element	Description
<splunk-control-group>	Wrapper element for a set of interface controls.
<splunk-select>	Selection control that takes options in HTML.
<splunk-radio-input>	Radio group that takes options in HTML.
<option>	Declares an option for select and radio group elements. Child element of these input elements.
<splunk-text-area>	Resizable text area.
<splunk-text-input>	Text input.
<splunk-color-picker>	Color picker element with three preconfigured palette types. Also allows a custom palette.
<splunk-color>	

Custom HTML element	Description
	Declares a color value for a custom color picker palette. Child element of the <code>&lt;splunk-color-picker&gt;</code> element.

These elements have styling consistent with standard Splunk Web elements, although they do not have the standard Splunk Web layout by default.

## Property namespacing

Use the following case-sensitive namespace element and syntax to specify a property in `formatter.html`.

```
{{VIZ_NAMESPACE}}.<property name>
```

The `{{VIZ_NAMESPACE}}` element is replaced with the namespaced portion of a property name. When the namespace element is combined with the property name, it works equivalently to the following fully-qualified syntax.

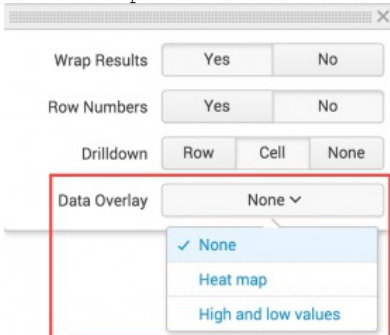
```
display.visualizations.custom.<app name>.<viz name>.<property name>
```

## Selection elements

The following elements present options and a selection interface to users.

### ***splunk-select***

Use the `<splunk-select>` element to create a selection control. Options appear in a list.



## Requirements

- Specify one or more `<option>` child elements defining the options that appear in the list.

### ***splunk-radio-input***

Use the `<splunk-radio-input>` element to declare a radio group control. There are three `<splunk-radio-input>` elements in this example.

## Requirements

- Specify one or more `<option>` child elements to indicate the options that appear as radio buttons.

### *option*

Use an `<option>` child element with a `<splunk-select>` or `<splunk-radio-input>` to specify available options. The `<option>` enclosed text appears as the option label. When a user selects an option, the `<option>` value is set as the control value.

This example shows an option list for specifying decimal precision. Each precision level is an `option`.

## Examples

### Example `splunk-select`

```
<splunk-select name="{{VIZ_NAMESPACE}}.legendPosition" value="right">
  <option value="right">Right</option>
  <option value="bottom">Bottom</option>
  <option value="left">Left</option>
  <option value="top">Top</option>
  <option value="none">None</option>
</splunk-select>
```

### Example `splunk-radio-input`

```
<splunk-radio-input name="{{VIZ_NAMESPACE}}.theme" value="light">
  <option value="light">Light</option>
  <option value="dark">Dark</option>
</splunk-radio-input>
```

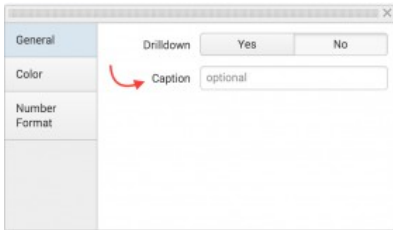
## Text entry elements

The following elements let users add text to a visualization.

### ***splunk-text-input***

Use a `<splunk-text-input>` element to create a text input control where users can enter up to a single line of text.

This example text input lets users specify a visualization caption.



### ***splunk-text-area***

Use a `<splunk-text-area>` element to create a text area control where users can type multiple lines of text.

## **Examples**

### **Example splunk-text-input**

```
<splunk-text-input name="{{VIZ_NAMESPACE}}.yAxisMaximum"></splunk-text-input>
```

### **Example splunk-text-area**

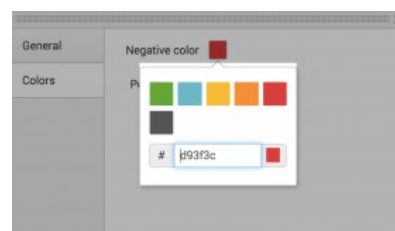
```
<splunk-text-area name="{{VIZ_NAMESPACE}}.xAxisTitle"></splunk-text-area>
```

## Color configuration element

This element lets users customize colors in a visualization.

### ***splunk-color-picker***

Use this element to provide a color configuration user interface.



## Guidelines

- Specify one of the following types for the color picker.
  - ◆ `splunkCategorical`. Default type if none is specified.
  - ◆ `splunkSemantic`
  - ◆ `splunkSequential`
  - ◆ `custom`
  - ◆ Specify colors in the custom palette or extend one of the available palette types using `<splunk-color>` tags.
- For custom color palettes, the `<splunk-color>` tag accepts valid CSS color strings. Invalid strings are ignored.
- You can use the `value` tag to set a default value for the picker.

## Example

```
<!-- Default color picker with splunkCategorical colors -->
<splunk-color-picker>
</splunk-color-picker>

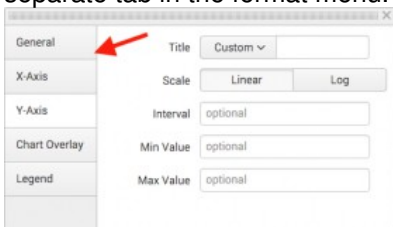
<!-- Color picker with splunkSemantic colors -->
<splunk-color-picker type="splunkSemantic">
</splunk-color-picker>

<!-- Custom color picker with only black and white. Default value is set to black -->
<splunk-color-picker type="custom" value="#000000">
  <splunk-color>#ffffff</splunk-color>
  <splunk-color>#000000</splunk-color>
</splunk-color-picker>

<!-- A splunkCategorical color picker that also includes black. Default value is set to black -->
<splunk-color-picker value="#000000">
  <splunk-color>#000000</splunk-color>
</splunk-color-picker>
```

## Format menu groupings and tabs

Developers can customize the format menu to group different form elements into sections. Each section renders as a separate tab in the format menu. This example shows a format menu with tabs for different format option groups.



## Guidelines

- Use the schema to implement multiple form elements at the top level of the HTML hierarchy.
- Each form element should have the class `splunk-formatter-section` and a `section-label` attribute to indicate the tab label.

### Example

```
<form class="splunk-formatter-section" section-label="Tab 1">
  ...
  <splunk-select>...</splunk-select>
  ...
</form>
<form class="splunk-formatter-section" section-label="Tab 2">
  ...
  <splunk-text-input>...</splunk-text-input>
  ...
</form>
```

## Wrapper for input elements and labels

### Input elements

Wrap all format menu input elements with this component.

```
<splunk-control-group>
```

You can specify the following `<splunk-control-group>` attributes.

Attribute	Description
label	Label for the input element. Appears in the UI.
help	String appearing underneath the control.

## Complete formatter.html example

This example includes a text input and a color picker.

```
<form class="splunk-formatter-section" section-label="Max value">
  <splunk-control-group label="Maximum dial value">
    <splunk-text-input name="{{VIZ_NAMESPACE}}.maxValue" value="100">
    </splunk-text-input>
  </splunk-control-group>
</form>
<form class="splunk-formatter-section" section-label="Dial color">
  <splunk-control-group label="Color">
    <splunk-color-picker name="{{VIZ_NAMESPACE}}.mainColor" value="#f7bc38">
    </splunk-color-picker>
  </splunk-control-group>
```



</form>

## Data handling guidelines

Handle search result data and errors in a custom visualization.

### Check for empty data

Whenever the search results change, `formatData` is called. Use this function to check for an empty results data object and throw an error.

#### Best practice

Show a helpful error message. For example, use the following message.

```
The search did not return any data.
```

#### Avoid

Do not display a blank page or dashboard panel.

---

### Check for invalid data

Handle cases in which the visualization search does not generate data in the correct format or type for rendering. Check for the expected data format or type before rendering the visualization.

#### Best practice

Show a helpful error message. For example, use the following message.

```
This visualization requires date or time information. Try using the timechart command in your query.
```

#### Avoid

Do not display the visualization when the data type or format is incorrect for rendering.

---

### Handle large data sets

Make sure that the custom visualization handles large result data sets correctly.

#### *Check for results that exceed configured row limits*

Check for results that exceed the configured results row limit. Compare the number of rows requested in the `visualization.js getInitialDataParams` function with the number of results that the search returns.

#### *Check search completion status*

In the case of long running searches, paging results are inaccurate before the search completes. As of the latest software version, you can use the `data.meta.done` boolean flag to check on search completion.

The `data.meta.done` flag is part of the data object that passes to `formatData` and `updateView` in `visualization.js`. When the flag is `true`, it indicates that the search is complete.

#### Example

This visualization code displays a message when the search completes.

```

...
updateView: function(data) {
  if (!data) {
    return;
  }
  var message = 'My search job is still running';
  if (data.meta.done) {
    message = 'My search job is done';
  }
  this.el.innerHTML = 'Status: ' + message;
}

```

### Best practice

Show a helpful warning message with the visualization in case there are more results than the visualization can render. For example, use the following message.

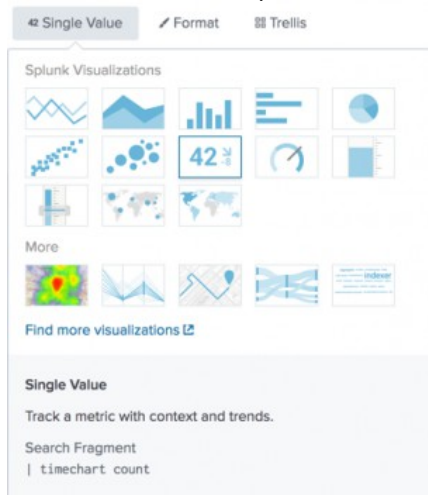
Warning: This visualization renders up to 10,000 data points. Results might be truncated.

## Design guidelines

To create a user experience that is consistent with standard Splunk visualizations, follow these guidelines.







### Visualization picker

Custom visualization preview icons, labels, and descriptions appear in the Visualization Picker user interface.



## Icon

Create a custom visualization preview icon. Follow these screenshot guidelines.

Recommendation	What to do	What to avoid
<b>preview.png</b>	Provide a preview.png file in the app directory.  preview.png	If no screenshot is available, a generic image appears.  Difficult to identify or find
<b>detail</b>	Use an appropriate detail level in the screenshot.  Detailed visualization preview	These visualization icons are difficult to understand.  Not enough detail Too much detail
<b>size</b>	Use an 116px by 76px image. Make sure that the image fills the available space fully.  No gaps or borders	Do not use a screenshot with gaps or borders.  Icon has gaps

## Description

Provide a visualization label and description to display in the Visualization Picker. List these attributes in the custom visualization `visualizations.conf` stanza.

## Example visualizations.conf stanza

```
[single]
label = Single Value
...
description = Track a metric with context and trends.
```

## Guidelines

### Character limits

Make sure that all strings in `visualizations.conf` follow these character limits before publishing the app.

- Label: 30 characters
- Description : 80 characters
- Search fragment: 80 characters

## Description best practices

Recommendation	What to do	What to avoid
Help users decide whether to use the visualization.	Tell users what they can do with the visualization.	Do not describe what the visualization looks like. <b>Avoid:</b> "This visualization has a circle with an arrow."

Recommendation	What to do	What to avoid
	<b>Best practice:</b> "Track a metric and trends over time."	
Use active voice to engage users.	Focus on a task that users can accomplish. <b>Best practice:</b> "Show how a metric varies across geographic regions."	Avoid technical descriptions of components. <b>Avoid:</b> "This visualization has a map that can be used with searches that generate an aggregate value."
Keep the description minimal.	Convey only the necessary information. <b>Best practice:</b> "Compare values or fields."	Do not repeat the visualization name. <b>Avoid:</b> "A color meter visualization shows a value in a range."

### Best practice examples

#### • Horizon chart

Use a baseline to show positive and negative changes for multiple time series.

#### • Real-time location tracker

Show physical asset locations in real time.

### Description template and terms

You can use the following components and term suggestions to create a custom visualization description.

Component	Suggested terms
<b>Action</b> What the user can do with the visualization	<ul style="list-style-type: none"> <li>• show</li> <li>• track</li> <li>• compare</li> <li>• plot</li> <li>• use</li> </ul>
<b>Information or behavior</b> What kinds of information or behavior the visualization shows	<ul style="list-style-type: none"> <li>• values</li> <li>• trends</li> <li>• metric</li> <li>• changes</li> <li>• relationships</li> <li>• fields</li> <li>• status</li> </ul>
<b>Presentation</b> How the visualization presents information or behavior	<ul style="list-style-type: none"> <li>• over</li> <li>• in</li> <li>• against</li> <li>• in relation to</li> <li>• between</li> <li>• using</li> </ul>
Key components	<ul style="list-style-type: none"> <li>• baseline</li> </ul>

Component	Suggested terms
Visualization component that adds meaning or emphasis	<ul style="list-style-type: none"> <li>• dataset</li> <li>• range</li> <li>• map</li> <li>• region</li> <li>• time</li> <li>• context</li> <li>• trend indicator</li> <li>• sparkline</li> <li>• icon</li> </ul>

## Text

Use the following settings for all chart axis, label, and legend text. For tooltip text guidelines, see [Tooltips](#).

### Font

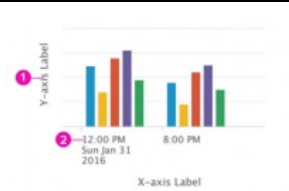
Lucida Grande typeface. Specify the CSS `font-family` property as shown here.

```
font-family: 'Lucida Grande', 'Lucida Sans Unicode', Arial, Helvetica
```

### Color

#3C444D (Dark gray)

### Label settings

	1. X- and Y-axis titles	2. Tick mark labels
	Line-height 16px  Size 12px	Line-height 12px  Size 11px







## Color

Use one of the following color palettes.

### *Semantic colors*











Use semantic colors to show meaning. For example, these colors can indicate value ranges in a results set.


Hex values	Palette
------------	---------

Hex values	Palette
[ '#DC4E41', '#F1813F', '#F8BE34', '#53A051', '#006D9C', '#3C444D' ]	 DC4E41  F1813F  F8BE34  53A051  006D9C  3C444D

### ***Categorical colors***

Categorical colors show how results belong to different categories.







Hex values	Palette
[ '#006D9C', '#4FA484', '#EC9960', '#AF575A', '#B6C75A', '#62B3B2', '#294E70', '#738795', '#EDD051', '#BD9872' ]	 006D9C  4FA484  EC9960  AF575A  B6C75A  62B3B2  294E70  738795  EDD051  BD9872
Hex values	Palette
[ '#5A4575', '#7EA77B', '#708794', '#D7C6B', '#339BB2', '#55672D', '#E6E1A', '#96907F', '#87BC65', '#CF7E60' ]	

Hex values	Palette
	 5A4575  7EA77B  708794  D7C6B  339BB2  55672D  E6E1A  96907F  87BC65  CF7E60
Hex values	Palette
[ '#7B5547', '#77D6D8', '#4A7F2C', '#F589AD', '#6A2C5D', '#AAABAE', '#9A7438', '#A4D563', '#7672A4', '#184B81' ]	 7B5547  77D6D8  4A7F2C  F589AD  6A2C5D  AAABAE  9A7438  A4D563  7672A4  184B81

Hex values	Palette

### ***Divergent colors***


Divergent colors emphasize high and low values in a results set. Shades for values between the maximum and minimum depend on the number of bins configured for results.

Hex values	Palette
[ '#236D9C', '#EC9960' ]	
[ '#62B3B2', '#AF575A' ]	236D9C  EC9960
[ '#62B3B2', '#AF575A' ]	62B3B2  AF575A
[ '#AF575A', '#F8BE34' ]	AF575A  F8BE34
[ '#F8BE34', '#4FA484' ]	F8BE34  4FA484
[ '#F8BE34', '#4FA484' ]	708794  5A4575
[ '#708794', '#5A4575' ]	294E70  B6C75A
[ '#294E70', '#B6C75A' ]	

### ***Sequential colors***

Sequential colors emphasize high values in a results set. The following hex values correspond to base colors for maximum values. Show minimum values using a lighter version of the base color.

Ensure that the minimum value appears in a visualization by using a value no lighter than 10% of the base color. Colors for values between the maximum and minimum are set according to the number of bins configured for results.

Hex values	Palette
[ '#1D92C5', '#D6563C', '#6A5C9E', '#31A35F', '#ED8440', '#3863A0' ]	



Hex values	Palette

## Layout

Proper spacing between tick marks, labels, and legends makes a visualization look clean and legible. Follow the spacing and margin guidelines shown here.



## Spacing

Between	Use this spacing
Y-axis label and visualization	10px
X-axis label and tick mark labels	10px
Visualization and legend	20px
Tick mark labels and visualization	5px

## Margin

Set a **15px** margin around the visualization panel.

## Chart elements

### Axis and gridline color

Keep gridlines and axis lines muted to maintain user focus on the data. Use the color settings shown here.



Gridlines  
#ebedef

Axis lines  
#d9dce0

---

### Legend swatch

Use a 16x12px swatch for each item in a legend.

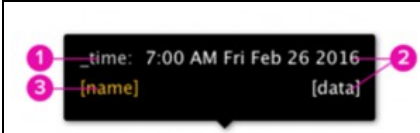
---

### Tooltips

#### Padding

10px

#### Text settings

	Size	Line-height	Color
	1. 12px	16px	#CCC
	2. 12px	16px	#FFF
	3. 12px	N/A	category color

#### Pointer position

Position the tooltip pointer at the center of any tooltip edge.

---

## Display size variation

Make sure that the custom visualization accommodates different display sizes.

### Guidelines

Scale horizontally when panel or window size changes

**Best practice:** Adjust all elements so that the visualization scales the whole width.

**Avoid:** Do not use fixed sizes for horizontal dimensions.

Implement a responsive design

**Best practice:** For small display widths, hide unnecessary labels and other elements.

---

## Custom visualizations in Simple XML

Include custom visualizations in Simple XML dashboards.

### Guidelines

#### Add a custom visualization to a panel

Use the app name and visualization name for the `<viz>` type value.

To add a custom visualization to a dashboard panel, indicate the visualization using the following syntax.

```
<viz type="[app_name.viz_name]">
...
</viz>
```

For example, add the `sample_viz` custom visualization to a dashboard panel.

```
<viz type="viz_sample_app.sample_viz">
...
</viz>
```

### ***Configure visualization properties in Simple XML***

Users can specify Simple XML option values for any visualization properties using this syntax:

```
<option name="[app].[visualization_name].[property_name]">[specified_value]</option>
```

For example, specify a maximum value for a custom visualization using this option. In this case, the app name is `viz_sample_app` and the visualization name is `sample_viz`.

```
<option name="viz_sample_app.sample_viz.maxValue">200</option>
```

---

## **Example Simple XML**

This example shows how to include and configure a custom visualization in a Simple XML dashboard panel.

In this example, the custom visualization app name is `viz_sample_app` and the visualization name is `sample_viz`.

```
<dashboard>
  <label>Sample</label>
  <row>
    <panel>
      <viz type="viz_sample_app.sample_viz">
        <search>
          <query>index=_internal | stats count</query>
        </search>
        <option name="viz_sample_app.sample_viz.maxValue">200</option>
        <option name="viz_sample_app.sample_viz.minValue">0</option>
      </viz>
    </panel>
  </row>
</dashboard>
```

---

## **Custom visualizations in SplunkJS**

Custom visualization components registered with the system are accessible from SplunkJS.

Visualizations published in an app can be used in SplunkJS dashboard extensions and SplunkJS pages. Apps can use their own visualizations in SplunkJS as well as visualizations from other installed apps. Custom visualizations that are built in the Splunk platform can be reused only within a Splunk software environment.

---

## Access and instantiate a visualization

Here are the steps for accessing and instantiating a visualization from a SplunkJS page.

1. Require the visualization registry, `splunkjs/mvc/visualizationregistry`.
  2. Call `visualizationregistry.getVisualizer(<app_name>, <visualization_name>)`. Use the name of the app that contains the visualization and the visualization name. The function returns a constructor.
  3. Use the constructor to instantiate the visualization. Pass it an `id`, `managerid`, and an `el`.
- 

### Example

This example SplunkJS page instantiates the `customViz` visualization from an app called `viz_sample_app`. The visualization renders in a `div` with the `id` `content`.

```
require([
  'jquery',
  'splunkjs/ready!',
  'splunkjs/mvc/visualizationregistry',
  'splunkjs/mvc/searchmanager'
],
function($, mvc, VisualizationRegistry, SearchManager){

  var customViz = VisualizationRegistry.getVisualizer('viz_sample_app', 'customViz');

  var mainManager = new SearchManager({
    id: 'mainManager',
    search: 'index = _internal | stats count'
  });

  var myViz = new customViz({
    id: 'myViz',
    managerid: 'mainManager',
    el: $('#content')
  }).render();
});
```

# Custom alert actions

## Custom alert actions overview

Unique use cases can require custom alerting functionality and integration.

Use the Splunk custom alert action API to create alert action apps that admins can download and install from Splunkbase. Users can access and configure installed custom alert actions in Splunk Web. The API lets you create a user experience consistent with the standard Splunk alerting workflow.

---

### Developer resources

Use the following resources to learn how to build a custom alert action.

API overview

- [Custom alert action component reference](#)

Build custom alert action components

- [Create custom alert configuration files](#)
- [Create a custom alert script](#)
- [Define a custom alert action user interface](#)
- [Optional custom alert action components](#)
- [Advanced options for working with custom alert actions](#)

Examples

- [Logger example](#)
- [HipChat example](#)
- [KV Store integration example](#)

Migration advice for script alert actions

- [Convert a script alert action to a custom alert action](#)
- 

### Additional resources

To try out a custom alert action, you can use the built-in webhook alert action to send notifications to a web resource, like a chat room or blog. For more information, see *Use a webhook alert action* in the *Alerting Manual*.

## Custom alert action component reference

Review required and optional custom alert action components and app directory structure.

### App directory structure

Here is the directory layout of an app that includes a custom alert action.

```
[app_name]
  appserver
```

```

static
    [app_icon].png
    [alternative_icon].png

bin
    [custom_alert_action_script]

default
    alert_actions.conf
    app.conf
    restmap.conf
    setup.xml
    data
        ui
            alerts
                [custom_alert_action].html

metadata
    default.meta

README
    alert_actions.conf.spec
    savedsearches.conf.spec

```

## App components

This app directory has the following components.

Component	File	Description	Required?
<b>Logic</b>	[custom_alert_action_script]	Alert action script or executable file	Yes
<b>User interface</b>	[custom_alert_action].html	HTML file defining the user interface for alert configuration	Yes
<b>Alert action configuration</b>	alert_actions.conf	Registers the custom alert action	Yes
<b>Spec files</b>	alert_actions.conf.spec	Declares alert action parameters	Optional
	savedsearches.conf.spec	Declares alert action parameters configured in the local savedsearches.conf file for the Splunk platform instance.	Optional
<b>App configuration</b>	app.conf	Defines app package and UI information	Yes
<b>Icons</b>	[app_icon].png	One or more icon image file(s)	Optional
<b>Setup</b>	setup.xml	Defines a UI for populating global settings at setup time	Optional
<b>Validation</b>	restmap.conf	Defines validation for parameters declared in savedsearches.conf	Optional
<b>Access control metadata</b>	default.meta	Defines alert action permission and scope	Optional

### Confidential information storage

Additionally, you can opt to use the password storage endpoint to store confidential information in an encrypted format. See [Confidential information storage](#).

## Set up custom alert configuration files

Learn how to define custom alert action app settings in configuration files.

### Custom alert action app configuration files

Here are all of the configuration files that you can use to manage a custom alert action app. Some files are required to make the app work and others are optional.

File	Description	Required?
<code>alert_actions.conf</code>	Contains settings for the custom alert action.	Yes
<code>app.conf</code>	Package and UI information about the app.	Yes
<code>savedsearches.conf</code>	Define instance settings for saved search actions.	A local copy is required on the Splunk platform instance but not in the custom alert action app directory.
<code>restmap.conf</code>	Define attribute/value pairs for REST endpoints and provide validation rules.	Optional
<code>alert_actions.conf.spec</code>	Describes attributes and possible values for configuring global saved search actions in <code>alert_actions.conf</code> .	Optional
<code>savedsearches.conf.spec</code>	Describes attributes and possible values for saved search entries in <code>savedsearches.conf</code> .	Optional
<code>default.meta</code>	Defines alert action permission and scope.	Optional

### Set up required configurations

#### ***alert\_actions.conf***

Create a stanza in `alert_actions.conf` to configure the custom alert action.

#### **Stanza naming**

Follow these guidelines when naming the alert action stanza.

- The stanza name must be unique. Two apps cannot define the same alert action.
- The stanza name can contain only the following characters.
  - ◆ alphanumeric characters
  - ◆ underscores
  - ◆ hyphens
- The stanza name cannot contain spaces.

Typically, developers name stanzas using lower case letters separated by underscores as needed. Once you have a stanza name, match the name of the script or executable file for the custom alert action to the stanza name.

#### **Alert action attributes**

The following attributes can be set in the alert action stanza within `alert_actions.conf`.

Attribute	Type	Default	Description
<code>is_custom</code>	boolean	0	

Attribute	Type	Default	Description
			Indicates if the app implements a custom alert action. Custom alert action developers should set this value to 1.
label	text	N/A	Display name of the alert action in the Splunk Enterprise UI.
icon_path	relative file path to the custom alert action icon. The icon appears in the Splunk Web user interface.		<p>To enable the custom alert action icon, indicate the relative path to the icon image file from</p> <p><code>\$\$SPLUNK_HOME\$/etc/[app]/appserver/static/.</code></p> <p>The best practice is to use a 48 x 48 px PNG file. The icon displays at 24 x 24 pixels.</p> <p>The custom alert action icon is not the same as the app icon that appears on Splunkbase. To use the Splunkbase app icon for the custom alert action icon in Splunk Web, specify <code>appIcon.png</code> as the <code>icon_path</code> value.</p>
alert.execute.cmd	text		Provide the name/path of the script or binary to invoke, especially to avoid conflicts for modular inputs and custom alert actions with the same name or scheme. If specifying a binary outside of the <code>[app]/bin</code> search path, use a <code>*.path</code> file, where the content of the file is the absolute path of the binary. Environment variables are replaced when reading path files.
alert.execute.cmd.arg.<n>	text		Change the command line arguments passed to the script when it is invoked.
payload_format	(xml   json)	xml	Indicates format for payload sent to STDIN of the user-provided script.
disabled	boolean	0	Indicates whether the alert action is disabled. Set to "1" to disable the alert action.
param.[param_name]			Custom alert action parameter that is passed to the script as part of the payload. All parameters in the alert action stanza are treated as custom settings for the custom alert action. They are all passed to the alert script as part of the XML or JSON configuration payload.

These additional settings from `alert_actions.conf` are also honored. For more details, see `alert_actions.conf`.

Setting	Type	Default	Description
command	search string to invoke	<pre>sendalert \$action_name\$ results_file="\$results.file\$" results_link="\$results.url"</pre>	<p>Partial search string executed by the scheduler when the alert is triggered. Developers can override default behavior to invoke a different custom search command or to pre-process the data before piping to <code>sendalert</code>.</p> <div> <p>If you plan to include a subsearch in your command search string, be aware that the Splunk software must run the subsearch as a separate search before it parses the rest of the search string, which puts a load on the search scheduler. Usage of the subsearch can reduce alert action performance and might cause fewer scheduled searches to be run successfully in your environment.</p> </div>

`maxtime` parameter for `command` search string "`$action.<action name>.maxtime{default=5m}$`" Sets the timeout for the command. Applies only to the `sendalert` command and custom search commands that use the deprecated Version 1 Custom Search Command protocol. Commands that use the Version 2 Custom Search Command protocol set their timeouts with the `maxwait` setting in `commands.conf`. Usage of either `maxtime` or `maxwait` are required, as they keep actions from backing up the alert queue and shutting down the search scheduler.

`maxinputs` parameter for `command` search string "`action.<action name>.maxresults{default=50000}`" Along with `max_action_results` in `limits.conf`, `maxinputs` sets the maximum number of inputs that can be passed to the command when it is



invoked. Applies only to streaming custom search commands that use the deprecated Version 1 Custom Search Command protocol. Use `max_action_results` for commands that use the Version 2 Custom Search Command protocol. `ttlinteger10p` Provides the minimum time to live, in seconds, of the search artifacts produced by this action in the dispatch directory. If the integer is followed by the letter 'p', this measures the minimum time to live in terms of scheduled periods rather than seconds. `hostname` Custom hostname. `forceCsvResult` true | false auto When set to auto, automatically detects if the `sendalert` command is in the search that the alert is based on or the `sendalert` command is used in the alert actions. If the `sendalert` command is detected (or if set to 'true'), the search results are stored in the CSV format in the `results.csv.gz` file in the dispatch directory. Otherwise the search results are stored in the default SRS format, which is a serialized Splunk-specific search results format.

## Example

The following example shows a stanza in the `alert_actions.conf` for a custom alert action.

**\$SPLUNK\_HOME\$etc/apps/[name]/default/alert\_actions.conf**

```
[logger]
is_custom = 1
label = My Alert Action
icon_path = myicon.png
payload_format = json
disabled = 0
# Custom params
param.foo = bar
param.param1 = I can use a token: $result.host$
savedsearches.conf
```

A local copy of `savedsearches.conf` captures alert action user configurations for a particular Splunk instance.

For each Splunk platform instance, `savedsearches.conf` user settings override any global `alert_actions.conf` alert action settings.

## Example

In this example, the `alert_actions.conf` file for a custom alert action defines a global parameter and setting for the alert action.

**alert\_actions.conf**

```
[my_custom_alert]
param.email_option = 0
```

In a Splunk platform instance, the following setting for the same parameter in the local `savedsearches.conf` file overrides the global setting from the app.

**savedsearches.conf**

```
action.my_custom_alert.param.email_option= 1
```

## How configurations propagate to the alert action

When the custom alert action script runs, it reads in payload information about the system and the alert. The payload includes alert action configurations merged from `alert_actions.conf` and `savedsearches.conf`.

The following example payload includes a `<configuration>` element with parameters and settings from the two files.

```

<alert>
  <server_host>localhost:8089</server_host>
  <server_uri>https://localhost:8089</server_uri>
  <session_key>1234512345</session_key>
  <results_file>
    /opt/splunk/var/run/splunk/12938718293123.121/results.csv.gz
  </results_file>
  <results_link>
    http://splunk.server.local:8000/en-US/app/search?sid=12341234.123
  </results_link>
  <sid>12341234.123</sid>
  <search_name>My Saved Search</search_name>
  <owner>admin</owner>
  <app>search</app>
  <configuration>
    <stanza name="[my_custom_alert]">
      <param name="[param_name_1]">[some value]</param>
      <param name="[param_name_2]">[other value]</param>
    </stanza>
  </configuration>
</alert>

```

- Note: For searches generated using the advanced search option, `results_file` and `results_link` parameters are not included in the payload passed to the custom alert action script.

## Optional configurations

For information on optional configuration files, see [Optional custom alert action components](#).

## Create a custom alert action script

### Alert action script workflow

The script executes the alert action, such as sending an email or connecting to a web resource. To execute the alert action, the script follows a workflow to get information about the triggered alert and run the alert action.

Typically, the script's workflow looks like this:

- Check the execution mode, based on command line arguments.
- Read configuration payload from `stdin`.
- Run the alert action.
- Terminate.

### *Executable files recognized for introspection*

There are several types of executable files recognized for introspection.

Recognized file types
<p><b>*Nix platforms</b></p> <ul style="list-style-type: none"> <li>• <code>filename.sh</code></li> <li>• <code>filename.py</code></li> </ul>

Recognized file types
<ul style="list-style-type: none"> <li>• <i>filename.js</i></li> <li>• <i>filename</i> (executable file without an extension)</li> </ul>
<b>Windows platforms</b> <ul style="list-style-type: none"> <li>• <i>filename.bat</i></li> <li>• <i>filename.cmd</i></li> <li>• <i>filename.py</i></li> <li>• <i>filename.js</i></li> <li>• <i>filename.exe</i></li> </ul>

## About the execution mode

When the alert action is triggered, the script receives one command line argument, which is the string `--execute`. This argument indicates the execution mode. Your script should check for the `--execute` argument. Additional execution modes might be added to this interface.

## About the script configuration payload

The `alert_actions.conf` file and `savedsearches.conf` file define the content of the configuration payload. Upon startup, the script reads the configuration from the payload. Developers typically create the configuration files before writing the script because of this dependency. The configuration file format is usually XML, but can be JSON if specified in `alert_actions.conf`.

The configuration payload contains:

- **Global information about the system**

- \* `splunkd` session key
- \* `splunkd` management URL

- **Information about the triggered alert and search**

- \* SID
- \* Saved search name
- \* Path to file containing the search results
- \* URL to the search results

- **Alert action configuration**

- \* This configuration contains the merged parameters of `alert_actions.conf` and `savedsearches.conf`.

- **The first search result**

## Script runtime threshold

The script runs separately for each triggered alert. It should have a brief execution time and terminate once the alert action execution completes. The script is forcefully terminated if the runtime exceeds its runtime threshold. The default runtime threshold is 5 minutes.

## Script naming guidelines

The name of the script should be the same as in its `alert_actions.conf` stanza. You can add an optional file name extension. For example, `myapp/bin/myalertaction.py` corresponds to `[myalertaction]` in `alert_actions.conf`. For more information, see [alert\\_actions.conf](#).

## Where to place the script or executable

Place the script or executable in the following directory:

`$SPLUNK_HOME/etc/apps/[myapp]/bin/`

### *Override a script with `alert.execute.cmd`*

Developers can use the `alert.execute.cmd` option to override the filename of the script to execute. You can use a custom binary and executed arguments for more flexibility. Create a stanza and place the path file and arguments in `alert_actions.conf`.

```
[myjavaaction]
...
alert.execute.cmd = java.path
alert.execute.cmd.arg.0 = -jar
alert.execute.cmd.arg.1 = $SPLUNK_HOME/etc/apps/myapp/bin/my.jar
alert.execute.cmd.arg.2 = --execute
```

## Script override considerations

- If you use a custom path file and arguments, make sure that the stanza name in `alert_actions.conf` is unique.
- If you use the `alert.execute.cmd` settings to specify a command to execute, the arguments are also overridden and not appended. `--execute` is not added unless manually specified,
- The external process starts with the arguments exactly as specified in the `alert_actions.conf` stanza.

### **.path file for a custom binary**

As shown in the example above, specify a `.path` file for `alert.execute.cmd` in the custom alert action's `alert_actions.conf` stanza. Absolute paths are not supported for `alert.execute.cmd`, although they can be used for its arguments. You can also use environment variables, such as `$SPLUNK_HOME` inside the `.path` file.

## Architecture-specific scripts

You can provide an architecture-specific version of a custom alert action script or executable by placing the appropriate version in the corresponding architecture-specific `/bin` directory for the app. Architecture-specific directories are available for these Intel-based architectures:

- Linux
- Apple (darwin)
- Windows

Only use a platform-specific directory when it is a requirement for that architecture. If you place a script in an architecture-specific directory, the script runs the appropriate version of the script. Otherwise, a platform-neutral version of the script runs in the default `/bin` directory.

### **`$$SPLUNK_HOME$etc/apps/[App]`**

```
/linux_x86/bin/[myscript]
/linux_x86_64/bin/[myscript]

/darwin_x86/bin/[myscript]
/darwin_x86_64/bin/[myscript]
```

### **`$$SPLUNK_HOME$etc\apps\[App]`**

```
\windows_x86\bin\[myscript]
\windows_x86_64\bin\[myscript]
```

## **Define a custom alert action user interface**

Add a custom alert action user interface to let users configure alert action properties. The following user interface API provides a user experience that is consistent with the Splunk platform.

### **File location**

Define the custom alert action interface in an HTML fragment file.

Place the HTML file in the following app directory location.

```
$$SPLUNK_HOME/etc/apps/[custom_alert_action_app_name]/default/data/ui/alerts/
```

### **Custom HTML elements**

The Splunk platform supports a set of custom HTML elements that manage the behavior and rendering of user interface controls. Here is an overview of the available elements.

Custom HTML element	Description
<code>&lt;splunk-control-group&gt;</code>	Wrapper element for a set of interface controls.
<code>&lt;splunk-search-dropdown&gt;</code>	Input control populated dynamically by a search. See <a href="#">Dynamic input controls</a> for more details.
<code>&lt;splunk-select&gt;</code>	Selection control that takes options in HTML.
<code>&lt;splunk-radio-input&gt;</code>	Radio group that takes options in HTML.
<code>&lt;option&gt;</code>	Declares an option for select and radio group elements. Child element of these input elements.
<code>&lt;splunk-text-area&gt;</code>	Resizable text area.
<code>&lt;splunk-text-input&gt;</code>	Text input.
<code>&lt;splunk-color-picker&gt;</code>	Color picker element with three preconfigured palette types. Also allows a custom palette.

Custom HTML element	Description
<code>&lt;splunk-color&gt;</code>	Declares a color value for a custom color picker palette. Child element of the <code>&lt;splunk-color-picker&gt;</code> element.

These elements have styling consistent with standard Splunk Web elements, although they do not have the standard Splunk Web layout by default.

## Wrapper for input elements and labels

Wrap all format menu input elements with this component.

```
<splunk-control-group>
```

You can specify the following `<splunk-control-group>` attributes.

Attribute	Description
<code>label</code>	Label for the input element. Appears in the UI.
<code>help</code>	String appearing underneath the control.

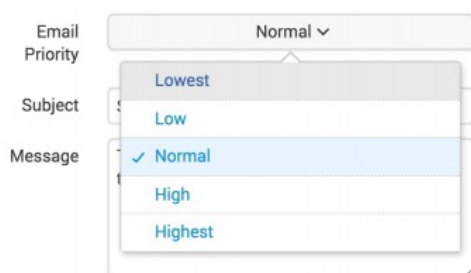
## Selection elements

The following elements present options and a selection interface to users.

### ***splunk-select***

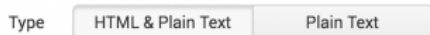
Use the `<splunk-select>` element to create a selection control. Options appear in a list.

Specify one or more `<option>` child elements defining available options. In this example, there are six email priority level options.



### ***splunk-radio-input***

Use the `<splunk-radio-input>` element to declare a radio group control.



## Requirements

Specify one or more `<option>` child elements to indicate the options that appear as radio buttons.

## ***option***

Use an `<option>` child element with a `<splunk-select>` or `<splunk-radio-input>` to specify available options. The `<option>` enclosed text appears as the option label. When a user selects an option, the `<option>` value is set as the control value.

## ***splunk-color-picker***

Use this element to provide a color configuration user interface.



- Specify one of the following color palette types for the color picker.
  - ◆ `splunkCategorical`. Default type if none is specified.
  - ◆ `splunkSemantic`
  - ◆ `splunkSequential`
  - ◆ `custom`
  - ◆ Specify colors in the custom palette or extend one of the available palette types using `<splunk-color>` tags.
- For custom color palettes, the `<splunk-color>` tag accepts valid CSS color strings. Invalid strings are ignored.
- You can use the `value` tag to set a default value for the picker.

Predefined color palettes are available as part of the custom alert action and custom visualization APIs. To learn more about the predefined color palette types, see [Color](#) in the [Design guidelines](#) for custom visualizations.

## **Text entry elements**

The following elements let users add custom text.

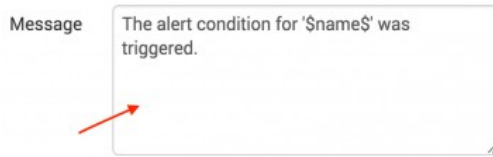
### ***splunk-text-input***

Use a `<splunk-text-input>` element to create a text input control. Users can enter up to a single line of text in a `splunk-text-input` control.



## ***splunk-text-area***

Use a `<splunk-text-area>` element to create a text area control. Users can enter multiple lines of text in a `<splunk-text-area>` control.



## **Input naming**

Input controls let users configure the namespaced parameters defined in the `savedsearches.conf` configuration file for the custom alert action.

Make sure that the input name matches the parameter name specified in `savedsearches.conf`. Matching the name ensures that user configurations propagate correctly to `savedsearches.conf`.

### **Example**

This example interface lets users specify the name of a chat room.

In `savedsearches.conf`, the `action.chat.param.room` setting specifies a chat room name.

```
# chat alert settings
action.chat.param.room = <string>
* Name of the room where notifications should go
* (required)
```

The user interface includes a text input for users to specify the chat room name. The input name matches the setting from `savedsearches.conf`.

```
<form>
  <splunk-control-group label="Chat room">
    <splunk-text-input "action.chat.param.room" id="chat_room">
      </splunk-text-input>
    </splunk-control-group>
  </form>
```

## **Dynamic input controls**

Add dynamically populated dropdown controls to a custom alert action interface. Use REST API, lookup table, or indexed data set search results to drive the dynamic input content.

### **Search to populate the input**

Consider the following details when writing a search to generate custom input options.



- In addition to provided platform commands and resources, you can use a custom search command and/or query a custom endpoint.
- For better performance, use a search that generates only the results that you need to populate the input. You can also consider commands to minimize processing.
- The search runs in the context of the current user and the deployment where the custom alert action is installed. When constructing the search, consider how dynamically populated options might vary depending on the resources available to the user and in the deployment.

### ***Dynamic input control attributes***

Use the following attributes to build a dynamically populated input dropdown.

Name	Description	Default	Required?
name	Input name. This name should match the setting name in <code>savedsearches.conf</code> to ensure that user configurations propagate from the input to the configuration file.	N/A	Yes
search	The query string to execute. Query the REST API, a lookup table, or indexed data.	N/A	Yes
label-field	Field name to use for dropdown option labels. Labels generated from this field are visible in the dropdown interface.	N/A	Yes
value-field	Field name to use for dropdown option values that correspond to the option labels.  In some cases, you can use the same results field for the <code>label-field</code> and <code>value-field</code> . In other cases, you might need to display human-readable labels from one field and use the corresponding values from another field. For example, an input might include a <code>user_name</code> field for the <code>label-field</code> and a <code>user_id</code> field for the <code>value-field</code> .	N/A	Yes
earliest	<code>earliest_time</code> in the search time range	" "	No
latest	<code>latest_time</code> in the search time range	"now"	No
app	App context in which the query runs. This specification can be useful when the search requires knowledge objects that are only available in a specific app context.	Defaults to the current app context.	No
allow-custom-value	Indicate whether to provide a field for the user to enter a custom value. Disabled by default. Developers can implement validation for user entered values.	false	No
max-results	Specify the maximum number of search results returned. Use any positive integer greater than 0.	1000	No

**Note:** Static or predefined options cannot be included in a dynamic dropdown input.

### ***Syntax and examples***

The following examples use queries against different resources to generate dropdown field labels and values.

## REST API

Use the `rest` search command to populate the input. You can query available `splunkd` endpoints or a custom endpoint.

```
<splunk-search-dropdown name="action.[alert_action_app_name].param.[alert_action_parameter]"
  search="| rest [endpoint path and optional parameters]"
  value-field="[results field for values]" label-field="[results field for labels]">
</splunk-search-dropdown>
```

### Example

This example queries the `services/data/indexes` endpoint and uses the `title` results field for option labels and values.

```
<splunk-control-group label="REST input">
  <splunk-search-dropdown name="action.controls_demo.param.search_dropdown"
    search="| rest /services/data/indexes"
    value-field="title" label-field="title">
  </splunk-search-dropdown>
</splunk-control-group>
```

### Lookup

Use a lookup table to populate the input.

```
<splunk-search-dropdown name="action.[alert_action_app_name].param.[alert_action_parameter]"
  search="| inputlookup [alert_action_lookup].csv"
  value-field="[results field for values]" label-field="[results field for labels]">
</splunk-search-dropdown>
```

### Example

This example searches a lookup table with geographical information. The input also includes a field for users to enter a custom value.

```
<splunk-control-group label="Allow custom values 1">
  <splunk-search-dropdown name="action.controls_demo.param.search_dropdown"
    search="| inputlookup geo_attr_countries.csv | search iso2=*
      | eval country=coalesce(country, iso2)"
    value-field="iso2" label-field="iso2"
    allow-custom-value>
  </splunk-search-dropdown>
</splunk-control-group>
```

### Indexed data

Search indexed data to populate the input.

```
<splunk-search-dropdown name="action.[alert_action_app_name].param.[alert_action_parameter]"
  search="index=[index_name] [...additional query content...]"
  earliest="[value]" latest=" [value]"
  value-field="[results field for values]" label-field="[results field for labels]">
</splunk-search-dropdown>
```

### Example

This example searches for internal data. It also sets a time range for the input.

```
<splunk-control-group label="Search driven dropdown 2">
```

```

<splunk-search-dropdown name="action.controls_demo.param.search_dropdown"
  search="index=_internal | streamstats count | table count | sort - count"
  earliest="-24h" latest="now"
  value-field="count" label-field="count">
</splunk-search-dropdown>
</splunk-control-group>

```

## Security considerations

Except for a dynamic dropdown control, only static HTML markup should be used in the interface. Do not include scripts or other constructs that could put your system at risk.

---

## Linking to static resources

To include URLs or links to static resources, use the replacement tag `{{ SPLUNKWEB_URL_PREFIX }}`.

## Optional custom alert action components

These items are optional, but you can add them to an app for additional functionality.

### Spec files

Create an `alert_actions.conf.spec` and/or a `savedsearches.conf.spec` file to describe new custom parameters in the `alert_actions.conf` or `savedsearches.conf` configuration files. Spec files are used for documentation and configuration file validation. Place spec files in a `README` directory within the app package.

For information on writing a spec file, see [Writing valid spec files](#). You can also see [Structure of a spec file](#). These topics address spec files for Modular Inputs, but are generally applicable for custom alert action apps.

### App setup

You can add a setup page to populate global configuration settings such as server addresses or credentials. A setup page is a page in your app that displays the first time your users launch the app. The setup page provides an interface in Splunk Web that allows your users to configure app settings.

For more information, see [Enable first-run configuration with setup pages in Splunk Cloud Platform or Splunk Enterprise on the Splunk Developer Portal](#).

### Metadata files

Use `default.meta` to define permissions and scope for alert actions. Typically you want to export the alert action globally. Here is an example configuration.

**`$SPLUNK_HOME$etc/apps/[custom_alert]/metadata/default.meta`**

```

[]
# Allow all users to read this app's contents.
# Allow only admin users to share objects into this app.
access = read : [ * ], write : [ admin ]

[alert_actions/logger]

```

```
# export actions globally
export = system
```

```
[alerts]
export = system
```

For more information, see the `default.meta.conf` reference in the *Admin* manual.

## Validation rules

Place validation rules for new parameters in `restmap.conf`.

These rules validate any new parameters and send error messages if validation rules are not met. Dynamic or external validation is not currently supported.

Here is an example of validation rules in `restmap.conf`.

```
[validation:savedsearch]
action.webhook.param.url = validate( match('action.webhook.param.url', "^https?://[^\s]+$"), "Webhook URL
is invalid")
```

For more information, see the `savedsearches.conf` and `restmap.conf` references in the *Admin* manual.

## Confidential information storage

To store confidential information such as passwords, API keys, or other credentials, you can use the app password storage endpoint, `storage/passwords`. This allows you to populate password storage entry via setup. Passwords are stored in encrypted form. You can use the `session_key` in the alert script to call back to `splunkd` and fetch cleartext information when the alert action is triggered.

For more information, see the `storage/passwords` endpoint documentation in the *REST API Reference Manual*.

- Note: Confidential information storage only works for setup-time configuration and does not work for instance settings created via the alert dialog in Splunk Web search user interface.

## Alert action icon file

You can include an icon file to represent the alert action separately from the app in Splunk Web. For example, users see the alert action icon in the dropdown menu for configuring an alert action. Place this icon file in the

`<app_name>/appserver/static` static assets directory along with the app icon file. Ensure that the alert stanza in `alert_actions.conf` includes an `icon_path` parameter that matches the icon file name. The best practice is to use a 48 x 48 px PNG file. The icon displays at 24 x 24 pixels.

The custom alert action icon is not the same as the app icon that appears on Splunkbase. To use the Splunkbase app icon for the custom alert action icon in Splunk Web, specify `appIcon.png` as the `icon_path` value.

It is recommended to name this icon file after the alert action. For example, you can use `my_alert_action_icon.png`.

## Convert a script alert action to a custom alert action

The run a script alert action is officially deprecated. Learn how to migrate existing alert action scripts to the custom alert action framework.

## Migration planning

- Start the migration process by comparing the scripted and custom alert action frameworks.
- To replicate scripted action functionality, you can use configuration files in combination with a custom alert action script. Review [Accessing script argument values in a custom alert action](#).
- Design a user interface to let users configure the alert action.

See the [Custom alert actions overview](#) for a complete guide to the custom alert action developer documentation.

## Framework comparison

Feature	Script alert action	Custom alert action	Notes
Numbered arguments for accessing alert values For example, use <code>\$0</code> or <code>SPLUNK_ARG_0</code> for the script name.	Yes	No	For custom alert actions, use configuration file parameters to access and pass values to the configuration payload that the alert action receives.
User interface API	No	Yes	The custom alert action API lets you build a configuration user interface that users can access when creating or editing an alert.
Configuration files	No	Yes	Custom alert actions use <code>alert_actions.conf</code> and <code>savedsearches.conf</code> for registration and configuration. They also use <code>app.conf</code> for app package and UI configuration.
Additional components	No	Yes	Custom alert action app components can include a <code>default.meta</code> file to specify permissions and access for the alert action. You can also include an icon for the alert action.

## Accessing script argument values in a custom alert action

One of the primary differences between the scripted alert action and custom alert actions is the API for accessing contextual values. These values are used in alert action configuration, dispatch, and communication.

The scripted alert action framework offers predefined positional arguments to access specific values.

In custom alert actions, a script can read in parameters and values from configuration files. Some of these values are available by default in the configuration payload. You can create configuration parameters in `alert_actions.conf` to pass additional values in to the payload. Developers typically set up configuration files before writing a custom alert action script.

For more information on working with configuration files and the configuration payload, see [Set up custom alert configuration files](#).

SPLUNK_ARG environment variable	Value	How to access the value in a custom alert action
---------------------------------	-------	--

0	Script name	The custom alert action script name must match the custom alert action stanza name in <code>alert_actions.conf</code> . It must also match the stanza name in <code>alert_actions.conf.spec</code> .
1	Number of events returned	Not available by default in the configuration payload. You can create a <code>param</code> to capture this search property using the <code>\$job.resultCount\$</code> token in <code>alert_actions.conf</code> .
2	Search terms	Not available by default in the configuration payload. You can create a <code>param</code> to capture this search property using the <code>\$job.request.search\$</code> token in <code>alert_actions.conf</code> .
3	Fully qualified search string	Not available by default in the configuration payload. You can create a <code>param</code> to capture this search property using the <code>\$job.request.qualifiedSearch\$</code> token in <code>alert_actions.conf</code> .
4	Name of saved search	Available as <code>search_name</code> key value pair in the custom alert action payload.
5	Trigger reason	Not available by default in the configuration payload.  To replicate the scripted alert trigger reason, create a <code>param</code> for a trigger reason string in <code>alert_actions.conf</code> . Use the <code>\$relation\$</code> and <code>\$quantity\$</code> job properties to concatenate the trigger reason string.
6	Results link	Available as <code>results_link</code> key value pair in the custom alert action configuration payload.
7	Not used in scripted alert actions	N/A
8	File in which the results for the search are stored.  Contains raw results in gzip file format.	Available as <code>results_file</code> key value pair in the custom alert action configuration payload.

## Building a user interface

Custom alert action apps include a user interface, while scripted actions do not. You can create a simple user interface as part of migrating an alert action script to a custom alert action. See [Define a custom alert action interface](#) for more information.

## Example

This example shows you how to migrate a scripted alert action that posts a notification to a Slack chat room.

The files referenced are examples similar to files that you might have in your Splunk platform `/scripts` and `etc/apps` directories.

### *Scripted alert action*

This `sendSlackAlertPythonOnly` script is in the `/bin/scripts` directory on the Splunk platform instance. This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

```

from __future__ import print_function
from builtins import str
import os
import sys
import requests

def send_slack(args_tuple, label=""):
    message = create_slack_msg(args_tuple, label)
    payload = str({"text": message})
    slackUrl = "https://hooks.slack.com/services/T41234fF2L/B4Z46756FW/Cowjj4NATJV7zqczOV23qWog"
    r = requests.post(slackUrl, data = payload)

def create_slack_msg(args_tuple, label=""):
    message = ""
    *This is a Splunk %s :horse:. Below are the 8 command line*
    *arguments used when invoking this scripted alert action.*
    1) Number of Events Returned: %s
    2) Search Terms: %s
    3) Fully Qualified Search String: %s
    4) Name of Saved Search: %s
    5) Trigger Reason: %s
    6) Linked To Saved Search: %s
    7) Tags or File Name for Results: %s
    8) File Name for Results (or none if no tags): %s
    "" % ((label,) + args_tuple)
    return message

if __name__ == "__main__":
    print("you_are_here", file=sys.stderr)
    send_slack(tuple(sys.argv[1:]), "Scripted Alert Action (simple python scripted alert action)")

```

### Comparable bash script example

The following bash script is an example of similar scripted alert action functionality in a shell script.

```

#!/bin/bash

slackUrl = "https://hooks.slack.com/services/T41234fF2L/B4Z46756FW/Cowjj4NATJV7zqczOV23qWog"

read -d "" message <<- EOF
*This is a Splunk Scripted Alert (simple bash script) :horse:. Below are the 8 commandline arguments*
*that Splunk invokes this scripted alert with:*
    1) Number of Events Returned: $1
    2) Search Terms: $2
    3) Fully Qualified Search String: $3
    4) Name of Saved Search: $4
    5) Trigger Reason: $5
    6) Linked To Saved Search: $6
    7) Tags or File Name for Results: $7
    8) File Name for Results (or none if no tags): $8
EOF

```

```
curl -X POST -H 'Content-type: application/json' --data '{"text": '$message'}' "$slackUrl"
```

### **Migrated custom alert action files**

The following files are part of a custom alert action app package in the `/etc/apps/alert_slack` directory. This custom alert action migrates the `sendSlackAlertPythonOnly` scripted alert action.

The example files here are only those required for migrating the scripted alert action. They do not represent all of the files in a custom alert action app. See the [Custom alert action component reference](#) for complete details.

## Script

This script uses symlinking to import functionality from the `sendSlackAlertPythonOnly` scripted alert action. Symlinking is optional when migrating a scripted alert action. This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

```
from __future__ import print_function
from future import standard_library
standard_library.install_aliases()
import sys
import os
import json
from urllib.parse import urlencode
import urllib.request, urllib.error, urllib.parse
import requests

# symlink this
from sendSlackAlertPythonOnly import create_slack_msg

def send_slack(settings):
    try:
        config = settings['configuration']
        args_tuple = (
            config['result_count'],
            config['search_query'],
            config['search_query'],
            settings['search_name'],
            config['trigger_reason'],
            settings['results_link'],
            'NA',
            settings['results_file']
        )
        message = create_slack_msg(args_tuple, "Custom Alert Action (that invokes a legacy, python, scripted alert)")
        payload = '{"text": '%s'}' % message
        slack_url = config['slack_url']
        res = requests.post(slack_url, data = payload)
        if 200 <= res.status_code < 300:
            print("DEBUG receiver endpoint responded with HTTP status=%d" % res.status_code,
                file=sys.stderr)
            return True
        else:
            print("ERROR receiver endpoint responded with HTTP status=%d" % res.status_code,
                file=sys.stderr)
            return False
    except Exception as e:
        print("ERROR Error %s" % e, file=sys.stderr)
        return False

if __name__ == "__main__":
    if len(sys.argv) < 2 or sys.argv[1] != "--execute":
        print("FATAL Unsupported execution mode (expected --execute flag)", file=sys.stderr)
        sys.exit(1)
    else:
        settings = json.loads(sys.stdin.read())
        if not send_slack(settings):
            print("ERROR Unable to contact slack endpoint", file=sys.stderr)
            sys.exit(2)
        else:
            print("DEBUG slack endpoint responded with OK status", file=sys.stderr)
```



## app.conf

```
[ui]
is_visible = 0
label = Log Slack Event Alert Action

[launcher]
author = Splunk
description = Log Slack Event Alert Action
version=6.6.0

[install]
state = enabled
is_configured = 1
```

## alert\_actions.conf

```
[slackcustomalert]
is_custom = 1
label = Slack Custom Alert Action
description = Send splunk event data to a Slack team room
icon_path = slacklogo.png
payload_format = json

param.trigger_reason = Saved Search [slackcustomalert] number of events($job.resultCount$)
param.result_count = $job.resultCount$
param.search_query = $job.search$
param.slack_url =
```

## alert\_actions.conf.spec

```
[slackcustomalert]

param.trigger_reason = <string>
* Provided for backwards compatibility with scripted alerts

param.result_count = <string>
* Number of results returned

param.search_query = <string>
* Search string
```

## savedsearches.conf.spec

```
# Slack event action settings

action.slackcustomalert.param.slack_url = <string>
* Slack chat room endpoint

action.slackcustomalert.param.trigger_reason = <string>
* Provided for backwards compatibility with scripted alerts

action.slackcustomalert.param.result_count = <string>
* Number of results returned

action.slackcustomalert.param.search_query = <string>
```

\* Search string

## User interface HTML file

```
form class="form-horizontal form-complex">
  <div class="control-group">
    <label class="control-label" for="slackcustomalert_slack_url">WebHook URL</label>
    <div class="controls">
      <textarea name="action.slackcustomalert.param.slack_url" id="slackcustomalert_slack_url"
style="width: 270px; max-width: 270px; height: 60px;"></textarea>
    </div>
  </div>
  <div class="control-group">
    <div class="controls">
      <span class="help-block" style="display: block; position: static; width: auto; margin-left: 0;">
        URL needed to POST data to your Slack team room
        <br />
        <a href="https://api.slack.com/incoming-webhooks" target="_blank"
          title="Slack Webhook Documentation">Slack Webhook Documentation<i
class="icon-external"></i></a>
      </span>
    </div>
  </div>
</form>
```

## Logger example for custom alert actions

The logger example implements a custom alert action that does the following:

- Creates a path to a log file when the alert first fires.
- Writes log messages to the log file when the alert fires.
- Writes log information to an existing Splunk Enterprise log file.

## Python file for logger example

logger.py implements custom alert actions. This script has been made cross-compatible with Python 2 and Python 3 using python-future.

**`$SPLUNK_HOME$etc/apps/logger_app/bin/logger.py`**

```
from __future__ import print_function
from builtins import str
import sys, os, datetime

def log(msg):
    f = open(os.path.join(os.environ["SPLUNK_HOME"], "var", "log", "splunk", "test_modalert.log"), "a")
    print(str(datetime.datetime.now().isoformat()), msg, file=f)
    f.close()

log("got arguments %s" % sys.argv)
log("got payload: %s" % sys.stdin.read())

print("INFO Hello STDERR", file=sys.stderr)
logger.py creates or updates a log file in the following location.
```

**`$SPLUNK_HOME$/var/log/splunk/test_modalert.log`**

The following is a sample of output generated by `logger.py` when an alert is triggered.

```
2015-03-07T01:41:42.430696 got arguments ['/opt/splunk/etc/apps/logger_app/bin/logger.py', '--execute']
2015-03-07T01:41:42.430718 got payload: <?xml version="1.0" encoding="UTF-8"?>
<alert>
  <app> logger_app </app>
  <owner>admin</owner>
  <results_file>/opt/splunk/var/run/splunk/dispatch/rt_scheduler__admin__
logger_app__RMD5910195c23186c103_at_1425692383_0.0/results.csv.gz</results_file>
  <results_link>http://myserver:8000/app/logger_app/@go?sid=rt_scheduler__admin__
logger_app__RMD5910195c23186c103_at_1425692383_0.0</results_link>
  <server_host>myserver</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>OCmOZHf37O^9fDktTrvNc6Kidz^68zs0Y7scufwRo6Lpdi5ZGmtxsPbIUlUKtjt9ZPG7gKz4Dq8
_eVntQ5EGR^N9rqkmg1dREAp8FFCduDwwvl6pEXEB^4w3MS6suwp9acw7JOlb</session_key>
  <sid>rt_scheduler__admin__ logger_app__RMD5910195c23186c103_at_1425692383_0.0</sid>
  <search_name>my_saved_search</search_name>
  <configuration>
    <stanza name=" my_saved_search"/>
  </configuration>
</alert>
```

## Configuration files for the logger example

The logger example for custom alert actions contains the following configuration files.

File	Description
alert_actions.conf	Define the properties of the custom alert action.
app.conf	Package and UI information about the add-on.  Required to display information about logger alert actions on the Alert Actions Manager page.

### ***alert\_actions.conf***

Defines the properties of the custom alert action.

Place the properties in a stanza with the base name of the script that implements the alert actions.

**`$SPLUNK_HOME/etc/apps/logger_app/default/alert_actions.conf`**

```
[logger]
is_custom = 1

#By default, custom alert actions are enabled
#disabled = 1

# The label, description, and icon appear in the alert
# actions dialog when a user configures an alert action
label = Log alert action
description = Custom action for logging fired alerts
icon_path = logger_logo.jpg
```

## ***app.conf***

Defines properties that appear in the Alert Actions Manager page.

```
[ui]
is_visible = 1
label = Mod Alert Tests

[launcher]
author = Splunk
description = Quick examples for testing mod alerts
version = 1.0

[install]
state = enabled
is_configured = 1
```

## **HTML file for the custom alert action form**

The HTML file defines the form elements for the custom alert action in the Splunk Enterprise UI. Best practice is to use markup consistent with the markup provided by Bootstrap. Bootstrap is a free collection of tools that contains HTML and CSS-based design templates.

The base name of the HTML file is the same as the base name of script that implements the alert action.

**`$SPLUNK_HOME/etc/apps/logger_app/default/data/ui/alerts/logger.html`**

```
<form class="form-horizontal form-complex">
  <p>Write log entries for this action.</p>
</form>
```

## **Access the logger alert action from Splunk Web**

From the home page, select the gear icon next to **Apps** and browse for the logger custom alert action.

## **HipChat example for custom alert actions**

The HipChat example implements an alert action that does the following:

- Posts a message to a HipChat room.
- Writes log messages to a Splunk Enterprise log file.

When a user selects the HipChat alert actions, the user can select from various actions that are available.

## **Python file for the HipChat Example**

This script has been made cross-compatible with Python 2 and Python 3 using python-future.

**`$SPLUNK_HOME/etc/apps/hipchat_app/bin/hipchat.py`**

```
from __future__ import print_function
from future import standard_library
```

```

standard_library.install_aliases()
import sys, json, urllib.request, urllib.error, urllib.parse

def send_message(settings):
    print("DEBUG Sending message with settings %s" % settings, file=sys.stderr)
    room = settings.get('room')
    auth_token = settings.get('auth_token')
    base_url = settings.get('base_url').rstrip('/')
    fmt = settings.get('format', 'text')
    print("INFO Sending message to hipchat room=%s with format=%s" % (room, fmt), file=sys.stderr)
    url = "%s/room/%s/notification?auth_token=%s" % (
        base_url, urllib.parse.quote(room), urllib.parse.quote(auth_token)
    )
    body = json.dumps(dict(
        message=settings.get('message'),
        message_format=fmt,
        color=settings.get('color', "green")
    ))
    print('DEBUG Calling url="%s" with body=%s' % (url, body), file=sys.stderr)
    req = urllib.request.Request(url, body, {"Content-Type": "application/json"})
    try:
        res = urllib.request.urlopen(req)
        body = res.read()
        print("INFO HipChat server responded with HTTP status=%d" % res.code, file=sys.stderr)
        print("DEBUG HipChat server response: %s" % json.dumps(body), file=sys.stderr)
        return 200 <= res.code < 300
    except urllib.error.HTTPError as e:
        print("ERROR Error sending message: %s" % e, file=sys.stderr)
        return False

if __name__ == "__main__":
    if len(sys.argv) > 1 and sys.argv[1] == "--execute":
        payload = json.loads(sys.stdin.read())
        if not send_message(payload.get('configuration')):
            print("FATAL Failed trying to send room notification", file=sys.stderr)
            sys.exit(2)
        else:
            print("INFO Room notification successfully sent", file=sys.stderr)
    else:
        print("FATAL Unsupported execution mode (expected --execute flag)", file=sys.stderr)
        sys.exit(1)

```

## Configuration files for the HipChat example

The HipChat example for custom alert actions contains the following configuration files.

File	Description
alert_actions.conf	Define the properties of the custom alert action.
app.conf	Package and UI information about the add-on.  Required to display information about logger alert actions on the Alert Actions Manager page.
hipchat_alert_icon.png	Icon file for the alert action in the Splunk Enterprise UI.
alert_actions.conf.spec savedsearches.conf.spec	Configuration spec files describing settings in alert_actions.conf and savedsearches.conf.

## ***alert\_actions.conf***

`alert_action.conf` defines the properties of the custom alert action. It also defines parameters to the `hipchat.py` script.

**`$(SPLUNK_HOME)/etc/apps/hipchat_app/default/alert_actions.conf`**

```
[hipchat]
is_custom = 1
label = HipChat
description = Send HipChat room notifications
icon_path = hipchat_alert_icon.png
payload_format = json

# base URL and Auth token available from your HipChat installation
param.base_url = http://hipchat.splunk.com/v2/
param.auth_token = Hr9marGO3ywwCyZqsE9r91MAMExtFpJKsxCnptbx
```

- Note: The HipChat example does not override the `alert.execute.cmd` command. For more details, see [Override a script with alert.execute.cmd](#).

## ***app.conf***

Defines properties that appear in the Alert Actions Manager page.

```
[ui]
is_visible = 1
label = Mod Alert Tests

[launcher]
author = Splunk
description = Quick examples for testing mod alerts
version = 1.0

[install]
state = enabled
is_configured = 1
```

## ***PNG file for the custom alert action icon***

The height and width dimensions of the PNG file should be equal. A PNG files with dimensions of 48x48 pixels works best.

**`$(SPLUNK_HOME)/etc/apps/hipchat_app/appserver/static/hipchat_alert_icon.png`**

## ***Spec files for the custom alert action***

The README directory contains the spec files for custom alert actions.

## ***alert\_actions.conf.spec***

`alert_action.conf.spec` describes custom settings for the custom alert action. These settings are used across all instances.

**`$(SPLUNK_HOME)/etc/apps/hipchat_app/README/alert_actions.conf.spec`**

```
[hipchat]
```

```
param.base_url = <string>
* HipChat API base URL - adjust if you're using you own server on premise

param.auth_token = <string>
* HipChat OAuth2 token
* see https://www.hipchat.com/docs/apiv2/auth
savedsearches.conf.spec
```

`savedsearches.conf.spec` describes additional `savedsearches.conf` settings introduced by the custom alert actions. These are per-instance settings.

Reference the parameters listed here with controls in the form that implements the UI for custom actions. See [Configure the UI for custom actions](#).

### **\$SPLUNK\_HOME\$etc/apps/hipchat\_app/README/savedsearches.conf.spec**

```
# HipChat alert settings

action.hipchat = [0|1]
* Enable hipchat notification

action.hipchat.param.room = <string>
* Name of the room to send the notification to
* (required)

action.hipchat.param.message = <string>
* The message to send to the hipchat room.
* (required)

action.hipchat.param.message_format = [html|text]
* The format of the room notification (optional)
* Default is "html"
* (optional)

action.hipchat.param.color = [red|green|blue|yellow|grey]
* Background color of the room notification (optional)
* (optional)

action.hipchat.param.notify = [1|0]
* Notify users in the room
* Defaults to 0 (not notifying users in the room)
* (optional)

action.hipchat.param.auth_token = <string>
* Override Hipchat API auth token from global alert_actions config
* (optional)
```

## **HTML file for the custom alert action form**

The HTML file defines the form elements for the custom alert action in the Splunk Enterprise UI.

Highlights of the HTML code:

- Defines a set of controls to display in the form for the custom action.

- Uses pre-defined CSS styles to define the controls in the form.
- Uses `{{SPLUNKWEB_URL_PREFIX}}` to define paths to local resources. [TBD]

**`$$SPLUNK_HOME$etc/apps/hipchat_app/default/data/ui/alerts/hipchat.html`**

```
<form class="form-horizontal form-complex">
  <div class="control-group">
    <label class="control-label" for="hipchat_room">Room</label>

    <div class="controls">
      <input type="text" name="action.hipchat.param.room" id="hipchat_room" />
      <span class="help-block">
        The name of a HipChat room.
      </span>
    </div>
  </div>
  <div class="control-group">
    <label class="control-label" for="hipchat_message">Message</label>

    <div class="controls">
      <textarea name="action.hipchat.param.message" id="hipchat_message" />
      <span class="help-block">
        The chat message for the HipChat room.
        Include tokens to insert text based on search results.
        <a href="{{SPLUNKWEB_URL_PREFIX}}/help?location=learnmore.alert.action.tokens"
target="_blank"
        title="Splunk help">Learn More <i class="icon-external"></i></a>
      </span>
    </div>
  </div>
  <div class="control-group">
    <label class="control-label">Message Format</label>

    <div class="controls">
      <label class="radio" for="hipchat_message_format_plain">
        <input id="hipchat_message_format_plain" type="radio"
name="action.hipchat.param.message_format" value="plain" />
        Plain Text
      </label>
      <label class="radio" for="hipchat_message_format_html">
        <input id="hipchat_message_format_html" type="radio"
name="action.hipchat.param.message_format" value="html" />
        HTML
      </label>
    </div>
  </div>
  <div class="control-group">
    <label class="control-label" for="hipchat_color">Background Color</label>

    <div class="controls">
      <select id="hipchat_color" name="action.hipchat.param.color">
        <option value="">None</option>
        <option value="red">Red</option>
        <option value="green">Green</option>
        <option value="blue">Blue</option>
        <option value="grey">Grey</option>
      </select>
      <span class="help-block">Change the background of the hipchat message.</span>
    </div>
  </div>
</div>
```



```

<div class="control-group">
  <div class="controls">
    <label class="checkbox" for="hipchat_notify">
      <input type="checkbox" name="action.hipchat.param.notify" id="hipchat_notify" value="1"/>
      Notify users in the room
    </label>
  </div>
</div>
<div class="control-group">
  <label class="control-label" for="hipchat_auth_token">Auth Token</label>

  <div class="controls">
    <input type="text" name="action.hipchat.param.auth_token" id="hipchat_auth_token"
placeholder="Optional"/>
    <span class="help-block">Override the globally configured HipChat Auth Token for this
alert.</span>
  </div>
</div>
</form>

```

## Advanced options for working with custom alert actions

Learn how to use additional features of custom alert actions.

### Invoke a custom alert action from a search

You can invoke an alert action by name using the `sendalert` command as part of a search. For testing purposes, you might want to invoke an alert action directly from search. You can pipe your search to `sendalert` and pass in parameters.

Here is the `sendalert` syntax.

```
sendalert <action-name> [options]
```

- `<action-name>` refers to an alert action in either `alert_actions.conf` or `savedsearches.conf`.
- `[options]` allows you to pass in key-value arguments starting with `param.` Each `param.` argument is merged with the corresponding token from `alert_actions.conf`.

For more information about using this command, see `sendalert` in the *Search Reference*.

### Pass search result values to alert action tokens

You can pass search result values to different alert action tokens when you use `sendalert`.

There are several available custom alert action tokens.

Token	Description
<code>\$result.&lt;fieldname&gt;\$</code>	Any field value from the first row of the search results
<code>\$job.&lt;property&gt;\$</code>	Any search job property
<code>\$server.&lt;property&gt;\$</code>	Properties returned by the server info endpoint
<code>\$app\$</code>	Name of the app containing the search
<code>\$cron_schedule\$</code>	Cron schedule for the alert

Token	Description
\$description\$	Search description
\$name\$	Name of the search or alert
\$next_scheduled_time\$	The next time the scheduled search runs
\$owner\$	Owner of the search
\$results_link\$	Link to the search results
\$search\$	Actual search string
\$trigger_date\$	Date when alert was triggered
\$trigger_timeHMS\$	Formatted time when the alert was triggered
\$trigger_time\$	Trigger time in unix epoch
\$alert.severity\$	Alert severity level
\$alert.expires\$	Alert expiration time

Custom alert action tokens work similarly to tokens for email notifications. To learn more, see [Use tokens in email notifications](#).

### Example

As an example, you might want to search for login failure events. You can pass the search results and some informational text to the `param.message` key. Then, you can use the `$result.<field_name>$` token to hold the corresponding field's value from your search results.

Here is what your query would look like.

```
index=_internal component=UiAuth action=login status=failure | sendalert chat param.room="Security Team Room" param.message="Login failed for user: $result.user$"
```

In this case, `user` is the result field name.

After receiving search results showing an admin role, the value passed to the alert script might look like this.

```
param.message = "Login failed for user: admin"
```

---

## Access alert action script logs

Developers can access logs of the alert action script using the Alert Actions manager page. Any information that your script prints to `STDERR` will be treated as a log message. Message prefixes, such as `DEBUG`, `INFO`, `WARN`, or `ERROR`, are treated as the log level.

To review logs for an alert action, select **Settings>Alert actions**. This takes you to the Alert Actions manager page. Select **View log events** for your alert action.

Custom alert action logging is similar to modular input logging. For more information, see [Set up logging](#).

## KV Store integration for custom alert actions

### *Integrate custom alert actions with the KV Store*

Integrate custom alert actions with the KV Store to track state and implement complex workflows. Here are some example use cases for KV Store integration.

- **Alert queue for review and approval.** To defer immediate alert actions, use the KV Store as a queue for alert action requests. Send alert action parameters, metadata, or an invocation string to the KV Store. Admin or other authorized users can review and approve queued alert action requests.
- **Alert action throttling.** Use the KV Store to track and retrieve state, such as most recent alert actions or an alert action count. An alert action script with custom throttling logic can use state information to suppress or run alert actions.
- **Logic to create and update service tickets.** Use a custom alert action script to create or update service tickets when an alert triggers. The script can log alerts and ticket information in the KV Store. When a new alert triggers, the script can check the KV Store for ticket history on similar alerts. If a ticket already exists for an alert with similar properties, then the script can update the ticket. If no ticket exists, the script can file a new one.

### *Example code*

Here is a code selection from a KV Store custom alert action script. The example app updates one field in a KV Store record. This script has been made cross-compatible with Python 2 and Python 3 using python-future.

```
from __future__ import print_function
from future import standard_library
standard_library.install_aliases()

import sys
import json
import urllib.request, urllib.parse, urllib.error
import urllib.request, urllib.error, urllib.parse

def request(method, url, data, headers):
    """Helper function to fetch JSON data from the given URL"""
    req = urllib.request.Request(url, data, headers)
    req.get_method = lambda: method
    res = urllib.request.urlopen(req)
    return json.loads(res.read())

payload = json.loads(sys.stdin.read())

config = payload.get('configuration', dict())
collection = config.get('collection')
record_name = config.get('name')
field = config.get('field')
value = config.get('value')

# Build the URL for the Splunkd REST endpoint
url_tmpl =
'%(server_uri)s/servicesNS/%(owner)s/%(app)s/storage/collections/data/%(collection)s/%(name)s?output_mode=json'
record_url = url_tmpl % dict(
    server_uri=payload.get('server_uri'),
    owner='nobody',
    app=urllib.parse.quote(config.get('app') if 'app' in config else payload.get('app')),
```

```

    collection=urllib.parse.quote(collection),
    name=urllib.parse.quote(record_name))
print('DEBUG Built kvstore record url=%s' % record_url, file=sys.stderr)
headers = {
    'Authorization': 'Splunk %s' % payload.get('session_key'),
    'Content-Type': 'application/json'}

# Fetch the record from the kvstore collection
try:
    record = request('GET', record_url, None, headers)
    print("DEBUG Retrieved record:", json.dumps(record), file=sys.stderr)
except urllib.error.HTTPError as e:
    print('ERROR Failed to fetch record at url=%s. Server response: %s' % (
        record_url, json.dumps(json.loads(e.read()))), file=sys.stderr)
    sys.exit(2)

# Update the record with the user supplied field value
data = {field: value}
record.update(data)

print('INFO Updating kvstore record=%s in collection=%s with data=%s' % (
    record_name, collection, json.dumps(data)), file=sys.stderr)

# Send the updated record to the server
try:
    response = request('POST', record_url, json.dumps(record), headers)
    print('DEBUG server response:', json.dumps(response), file=sys.stderr)
except urllib.error.HTTPError as e:
    print('ERROR Failed to update record:', json.dumps(json.loads(e.read()))), file=sys.stderr)
    sys.exit(3)

```

# Modular inputs

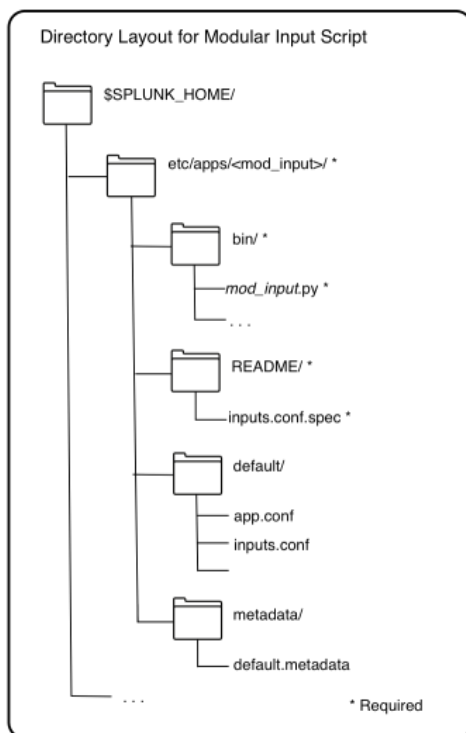
## Modular inputs basic example

This topic shows the steps necessary to create a modular input. It uses a trivial *Hello, World* style script that lets you concentrate on the basic framework and structure of modular inputs. It omits details of an actual script you might use to index a stream of data. It also omits advanced configuration data you might use to fine tune the operation of the modular input.

The example uses Python as a scripting language. However, you can use a scripting language of your choice to create the script. The script should contain the same functional parts that the example Python script illustrates. The Splunk Developer Portal contains modular input examples for each of the Splunk SDKs.

## Basic implementation requirements

A modular input is essentially a Splunk add-on. You place the modular input implementation in the same location you place apps and add-ons.



Directory	Description
<b>bin</b>	Required. Contains the script for the modular input.
<b>README</b>	Required. Contains <code>inputs.conf.spec</code> to register the modular input scheme.
<b>default</b>	Optional. Contains <code>app.conf</code> to configure the modular input as an add-on.
<b>metadata</b>	Optional. Contains <code>default.meta</code> to set permissions to share the script.

## Script modes

A script for a modular input typically runs in three modes: introspection, execution, and validation.

Script mode	Description
<b>Introspection</b>	<p>Defines the endpoints and behavior of the script. A modular input script must provide an introspection routine, even if it is a trivial routine that exits with a return code of zero.</p> <p>The script must define the command line argument, <code>--scheme</code>, to access the introspection routine.</p>
<b>Execution</b>	Streams data for indexing.
<b>Validation</b>	<p>Optional. Validates input data. If present, this routine guarantees that the script only accepts valid data.</p> <p>When implementing validation define the command line argument, <code>--validate-arguments</code>, to access the validation routine.</p>

## Essential Python script and configuration file for modular inputs

This minimal modular input contains a Python script file that creates a source type based on user inputs. The script contains an empty introspection routine and an empty validation routine. `hello_mi` is the name of the add-on that implements the modular input.

### Python script file

This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

```
# $SPLUNK_HOME/etc/apps/hello_mi/bin/hello.py
from __future__ import print_function
from builtins import str
import sys
import xml.dom.minidom, xml.sax.saxutils

# Empty introspection routine
def do_scheme():
    pass

# Empty validation routine. This routine is optional.
def validate_arguments():
    pass

# Routine to get the value of an input
def get_who():
    try:
        # read everything from stdin
```

```

config_str = sys.stdin.read()

# parse the config XML
doc = xml.dom.minidom.parseString(config_str)
root = doc.documentElement
conf_node = root.getElementsByTagName("configuration")[0]
if conf_node:
    stanza = conf_node.getElementsByTagName("stanza")[0]
    if stanza:
        stanza_name = stanza.getAttribute("name")
        if stanza_name:
            params = stanza.getElementsByTagName("param")
            for param in params:
                param_name = param.getAttribute("name")
                if param_name and param.firstChild and \
                    param.firstChild.nodeType == param.firstChild.TEXT_NODE and \
                    param_name == "who":
                    return param.firstChild.data
except Exception as e:
    raise Exception("Error getting Splunk configuration via STDIN: %s" % str(e))

return ""

# Routine to index data
def run_script():
    print("hello world, %s!" % get_who())

# Script must implement these args: scheme, validate-arguments
if __name__ == '__main__':
    if len(sys.argv) > 1:
        if sys.argv[1] == "--scheme":
            do_scheme()
        elif sys.argv[1] == "--validate-arguments":
            validate_arguments()
        else:
            pass
    else:
        run_script()

    sys.exit(0)

```

### ***Configuration file for modular inputs***

`inputs.conf.spec` defines the default scheme for the modular input. The configuration file must contain at least one stanza referencing the input. Each stanza must contain one or more parameters. The values for the parameters in the configuration file are not used.

```

*$SPLUNK_HOME/etc/apps/hello_mi/README/inputs.conf.spec

[hello://<default>]
*Set up the hello scheme defaults.

who = <value>

```

- **Note:** Avoid adding the `start_by_shell` parameter to `inputs.conf.spec`. This parameter should only be used in `inputs.conf`. See [Override default run behavior for modular input scripts](#) for more information.

## Access the modular input from Splunk Web

After creating the modular input, you can access it various ways from Splunk Web, and also from the Splunk Enterprise management port.

**Note:** Screen captures for this topic are from Splunk Enterprise 6. The layout from earlier versions may differ.

### Data inputs

Navigate to **Settings > Data inputs** to view the input under **Local inputs**.

The screenshot shows the 'Data inputs' page in Splunk Web. The navigation bar at the top includes 'splunk', 'Apps', 'Administrator', 'Messages', 'Settings', 'Activity', 'Help', and 'Find'. The page title is 'Data inputs'. Below the title, there is a section for 'Local inputs' with a description: 'Set up data inputs from files and directories, network ports, and scripted inputs. If you want to set up forwarding and receiving between two Splunk instances, go to [Forwarding and receiving](#).' Below this is a table with columns 'Type', 'Inputs', and 'Actions'.

Type	Inputs	Actions
<b>Files &amp; directories</b> Index a local file or monitor an entire directory.	5	<a href="#">Add new</a>
<b>TCP</b> Listen on a TCP port for incoming data, e.g. syslog.	0	<a href="#">Add new</a>
<b>UDP</b> Listen on a UDP port for incoming data, e.g. syslog.	0	<a href="#">Add new</a>
<b>Scripts</b> Run custom scripts to collect or generate more data.	1	<a href="#">Add new</a>
<b>hello</b>	3	<a href="#">Add new</a>

Below the table is a section for 'Forwarded inputs' with a table showing 'Windows Event Logs' with 0 inputs and an 'Add new' action.

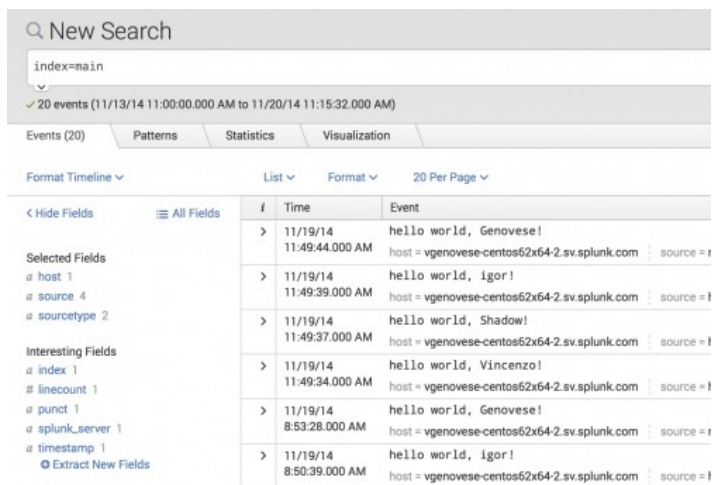
Click **Add new** to add additional data for your input.

The screenshot shows the 'Add Data' wizard in Splunk Web. The navigation bar at the top is the same as the previous screenshot. The page title is 'Add Data'. Below the title is a progress bar with three steps: 'Select Forwarders', 'Select Source', and 'Done'. The 'Select Source' step is currently selected. Below the progress bar is a list of input types: 'Files & Directories', 'TCP / UDP', 'Scripts', and 'hello'. The 'Files & Directories' input type is selected, and its configuration form is shown on the right. The form has two required fields: 'name' and 'who'. Below the form is a 'More settings' link with a checkbox.



## Search page

After creating the modular input and adding some data, create the following search from the Search page to see event listings from your modular input.



New Search

index=main

20 events (11/13/14 11:00:00.000 AM to 11/20/14 11:15:32.000 AM)

Events (20) Patterns Statistics Visualization

Format Timeline List Format 20 Per Page

< Hide Fields All Fields

Selected Fields

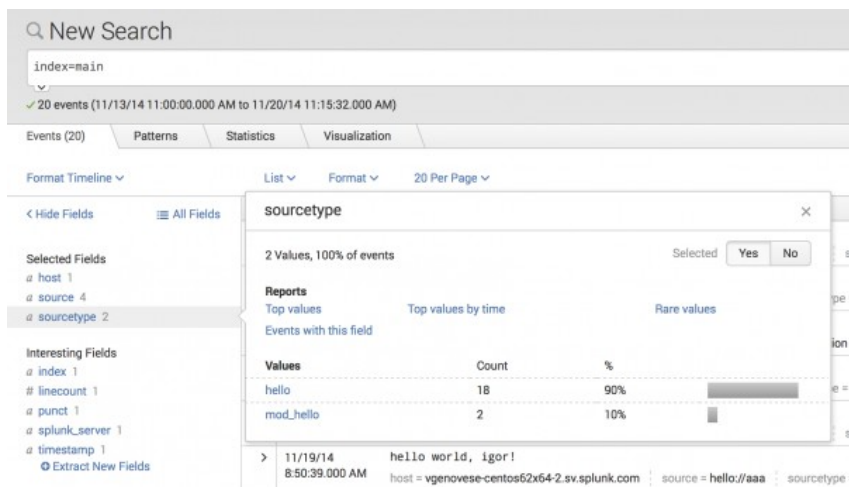
- host 1
- source 4
- sourcetype 2

Interesting Fields

- index 1
- linecount 1
- punct 1
- splunk\_server 1
- timestamp 1
- Extract New Fields

#	Time	Event
>	11/19/14 11:49:44.000 AM	hello world, Genovese! host = vgenovese-centos62x64-2.sv.splunk.com source = f
>	11/19/14 11:49:39.000 AM	hello world, igor! host = vgenovese-centos62x64-2.sv.splunk.com source = f
>	11/19/14 11:49:37.000 AM	hello world, Shadow! host = vgenovese-centos62x64-2.sv.splunk.com source = f
>	11/19/14 11:49:34.000 AM	hello world, Vincenzo! host = vgenovese-centos62x64-2.sv.splunk.com source = f
>	11/19/14 8:53:28.000 AM	hello world, Genovese! host = vgenovese-centos62x64-2.sv.splunk.com source = f
>	11/19/14 8:50:39.000 AM	hello world, igor! host = vgenovese-centos62x64-2.sv.splunk.com source = f

Click the **sourcetype** link to view details of the source types you created.



New Search

index=main

20 events (11/13/14 11:00:00.000 AM to 11/20/14 11:15:32.000 AM)

Events (20) Patterns Statistics Visualization

Format Timeline List Format 20 Per Page

< Hide Fields All Fields

Selected Fields

- host 1
- source 4
- sourcetype 2

Interesting Fields

- index 1
- linecount 1
- punct 1
- splunk\_server 1
- timestamp 1
- Extract New Fields

sourcetype

2 Values, 100% of events

Selected Yes No

Reports

- Top values
- Top values by time
- Rare values

Events with this field

Values	Count	%
hello	18	90%
mod_hello	2	10%

> 11/19/14 8:50:39.000 AM hello world, igor!  
host = vgenovese-centos62x64-2.sv.splunk.com source = hello://aaa sourcetype =

## Splunk Enterprise management port

You can access the REST endpoint for the modular input from the Splunk Enterprise management port. This example uses the default settings to access the REST endpoint:

`https://localhost:8089/servicesNS/admin/`



## Add introspection and validation routines

To enhance the basic implementation you can add introspection and validation routines. The examples in [Modular inputs examples](#) provide details on introspection and validation.

## Create modular inputs

This topic provides details on creating a modular input script, defining an introspection scheme, and the impact of enabling, disabling, and updating modular input scripts. It also covers overriding default modular input script run behavior for \*nix and Windows.

Other features regarding creating modular inputs, listed below, are covered elsewhere in this manual:

- [Set up logging](#)
- [Set up external validation](#)
- [Set up streaming \(simple or XML\)](#)
- [Modular inputs configuration](#)
  - ◆ [Create a modular input spec file](#)
- [Create a custom user interface](#)
- [Developer tools for modular inputs](#)

## Create a modular input script

A script that implements modular inputs runs in three scenarios:

1. Returns the introspection scheme to splunkd.  
The introspection scheme defines the behavior and endpoints of the script, as described in [Define a scheme for introspection](#). Splunkd runs the script to determine the behavior and configuration.
2. Validates the script's configuration.  
The script has routines to validate its configuration, as described in [Set up external validation](#).

### 3. Streams data.

The script streams event data that can be indexed. The data can be streamed as plain text or as XML, as described in [Set up streaming](#).

The following pseudo-code describes the behavior of a modular input script. This example assumes that there is a valid spec file, as described in [Modular inputs spec file](#). This also assumes that you are checkpointing data to avoid reading from the same source twice, as described in [Data checkpoints](#).

```
Define an introspection scheme
  Implement --scheme arg to print the scheme to stdout (scenario 1)
Implement routines to validate configuration
  Implement --validate-arguments arg to validate configuration (scenario 2)
  If validation fails, exit writing error code to stdout
Read XML configuration from splunkd
Stream data as text or as XML, using checkpoints (scenario 3)
  If checkpoint exists
    Exit
  Else
    While not done
      Write event data to stdout
    Write checkpoint
```

#### **Architecture-specific scripts**

Typically, you use the default bin directory for scripts:

```
$SPLUNK_HOME/etc/apps/<myapp>/bin/<myscript>
```

However, you can provide an architecture-specific version of a modular input script by placing the appropriate version of the script in the corresponding architecture-specific `bin` directory in your Splunk Enterprise installation.

Architecture-specific version directories are only available for the following subset of architectures that Splunk Enterprise supports. The architecture-specific directories are all Intel-based.

- Linux
- Windows
- Apple (darwin)

The following `bin` directories, relative to `$SPLUNK_HOME/etc`, are available for the corresponding Intel architectures:

```
/apps/<myapp>/linux_x86/bin/<myscript>
/apps/<myapp>/linux_x86_64/bin/<myscript>

\apps\<myapp>\windows_x86\bin\<myscript>
\apps\<myapp>\windows_x86_64\bin\<myscript>

/apps/<myapp>/darwin_x86/bin/<myscript>
/apps/<myapp>/darwin_x86_64/bin/<myscript>
```

If you place a script in an architecture-specific directory, the script runs the appropriate version of the script if installed on that platform. Otherwise, a platform-neutral version of the script runs in the default `bin` directory.

**Note:** Always have a platform-neutral version of the script in the default `bin` directory. Only use a platform-specific directory if required for that architecture.

## ***Executable files recognized for introspection***

The following type of executable files are recognized for introspection:

- \*Nix platforms  
    filename.sh  
    filename.py  
    filename (executable file without an extension)
- Windows platforms  
    filename.bat  
    filename.cmd  
    filename.py  
    filename.exe

## ***Example scripts***

See [Modular Inputs examples](#) for listings and descriptions of Modular Inputs example scripts. It contains the following examples:

[Twitter example](#)  
[Amazon S3 example](#)

## ***General tips on writing scripts***

The Build scripted inputs topic provides the section [Writing reliable scripts](#) that contains some tips and best practices for writing scripts.

## **Define a scheme for introspection**

You define both the behavior and endpoints for a script in an XML scheme that the script returns to splunkd.

During introspection, splunkd reads the scheme to implement your script as a modular input. Introspection determines the following:

- The endpoint definition for your script, which includes required and optional parameters to create and modify the endpoint.
- The title and description for the script, which is used in the Settings pages for creating or editing instances of the script.
- Behavior for the script such as:
  - ◆ Streaming in XML or plain text
  - ◆ Use a single script or multiple script instances
  - ◆ Validate your scheme configuration

## ***Introspection defaults***

Providing an introspection scheme with your script is optional.

If you do not provide the introspection scheme, the Settings page displays default values, which may or may not be

appropriate for your script.

If you do provide an introspection scheme, each element in the scheme is optional. If you do not provide an element, then Splunk software uses the default value for that element, which may or may not be appropriate for your script.

Your script must provide a "--scheme" argument, which when specified, does the following:

- If you implement an introspection scheme, writes the scheme to stdout.
- If you do not provide an introspection scheme, exits with return code 0. Splunk software uses the default introspection scheme in this scenario.

**Note:** When getting started writing scripts, consider using the default introspection scheme. You only need to write your own introspection scheme to specify behavior that differs from the default behavior.

### ***Example scheme***

The following snippet from a script contains an example XML scheme. It also contains snippets that show the routines to return the scheme for splunkd introspection. The introspection scheme must be UTF-8 encoded.

**Note:** See also [Introspection scheme](#) and [Splunk Manager pages](#) to view how this scheme affects the display in Splunk Web.

### **XML scheme snippets**

```
. . .
SCHEME = "<?xml version='1.0'><scheme>
  <title>Amazon S3</title>
  <description>Get data from Amazon S3.</description>
  <use_external_validation>true</use_external_validation>
  <streaming_mode>xml</streaming_mode>

  <endpoint>
    <args>
      <arg name="name">
        <title>Resource name</title>
        <description>An S3 resource name without the leading s3://.
          For example, for s3://bucket/file.txt specify bucket/file.txt.
          You can also monitor a whole bucket (for example by specifying 'bucket'),
          or files within a sub-directory of a bucket
          (for example 'bucket/some/directory/'; note the trailing slash).
        </description>
      </arg>

      <arg name="key_id">
        <title>Key ID</title>
        <description>Your Amazon key ID.</description>
      </arg>

      <arg name="secret_key">
        <title>Secret key</title>
        <description>Your Amazon secret key.</description>
      </arg>
    </args>
  </endpoint>
</scheme>
```

```

"""
. . .
def do_scheme():
    print SCHEME
. . .
if __name__ == '__main__':
    if len(sys.argv) > 1:
        if sys.argv[1] == "--scheme":
            do_scheme(). . .

```

### Introspection scheme details

Use <scheme> tags to define an introspection scheme. <scheme> can contain the following top-level elements:

#### Top-level elements for introspection <scheme>

Tag	Description
<title>	Provides a label for the script. The label appears in the Settings page for <b>Data inputs</b> .
<description>	Provides descriptive text for title in the Settings page for <b>Data inputs</b> . The description also appears on the <b>Add new data inputs</b> page.
<use_external_validation>	true   false. (Default is false.) Enables external validation.
<streaming_mode>	xml   simple (Default is simple, indicating plain text.) Streams inputs as xml or plain text.
<use_single_instance>	true   false (Default is false.) Indicates whether to launch a single instance of the script or one script instance for each input stanza. The default value, false, launches one script instance for each input stanza.
<endpoint>	Contains one or more <arg> elements that can be used to change the default behavior that is defined in the <code>inputs.conf.spec</code> file.  The parameters to an endpoint are accessible from the management port to Splunk Enterprise. Additionally, Splunk Web uses the endpoint to display each <arg> as an editable field in the <b>Add new data inputs</b> Settings page.  See below for details on specifying <endpoint>.

The <endpoint> element allows you to modify the default behavior that is defined in the `inputs.conf.spec` file. The following table lists the child elements to <endpoint>:

#### Details for the <endpoint> element

Tag	Description
<args>	Can contain one or more <arg> elements, defining the parameters to an endpoint.
<arg>	Defines the details of a parameter. <arg> can contain the following elements:  <div style="margin-left: 40px;"> &lt;title&gt;  &lt;description&gt;  &lt;validation&gt; </div>

Tag	Description
	<code>&lt;data_type&gt;</code> <code>&lt;required_on_edit&gt;</code> <code>&lt;required_on_create&gt;</code>
<code>&lt;title&gt;</code>	Provides a label for the parameter.
<code>&lt;description&gt;</code>	Provides a description of the parameter.
<code>&lt;validation&gt;</code>	<p>Define rules to validate the value of the argument passed to an endpoint create or edit action. See <a href="#">Validation of arguments</a> for details.</p> <p>You can also perform a higher level validation on a script, using the <code>&lt;use_external_validation&gt;</code> tag. See <a href="#">Set up external validation</a> for more information.</p>
<code>&lt;data_type&gt;</code>	<p>Specify the data type for values returned in JSON format.</p> <p>Splunk endpoints can return data in either JSON or Atom (XML) format. To handle data returned in JSON format, use <code>&lt;data_type&gt;</code> to properly define the datatype for the streamed data. Default datatype is string.</p> <p>Valid values are:</p> <pre> string number boolean </pre> <p>This has no effect for data returned in Atom format. New to Atom? For an introduction go to <a href="#">AtomEnabled.org</a>.</p>
<code>&lt;required_on_edit&gt;</code>	<p>true   false (Default is false.)</p> <p>Indicates whether the parameter is required for edit. Default behavior is that arguments for edit are optional. Set this to true to override this behavior, and make the parameter required.</p>
<code>&lt;required_on_create&gt;</code>	<p>true   false (Default is true.)</p> <p>Indicates whether the parameter is required for create. Default behavior is that arguments for create are required. Set this to false to override this behavior, and make the parameter optional.</p>

### **Built-in arguments and actions**

There are several arguments and actions that are always supported by a modular input endpoint.

The following arguments are implicit, and do not need to be defined in an introspection scheme:

```

source
sourcetype
host
index
disabled
interval
persistentQueue
persistentQueueSize
queueSize

```

The following actions are also implicit, and do not need to be defined in an introspection scheme:

enable/disable

Disabling an item shuts down a script. Enabling starts it up.

reload

Works on the endpoint level. Scripts that handle all of the enabled input stanzas are restarted.

### Validation of arguments

Use the `<validation>` tag to define validation rules for arguments passed to an endpoint create or edit action. This allows you to provide input validation for users attempting to modify the configuration using the endpoint.

For example, the following validation rule tests if the value passed for the argument is a boolean value:

```
<arg name="myParam">
  <validation>is_bool('myParam')</validation>
  . . .
</arg>
```

You can specify a validation rule for each `arg`, as shown in the above example for the `myParam` argument. The parameter passed to the function must match the name of the argument.

The Splunk platform provides built-in validation functions that you can use. `param` in each function must match the name specified for `<arg>`.

Validation function	Description
<code>is_avail_tcp_port(param)</code>	Is the value a valid port number, available for TCP listening.
<code>is_avail_udp_port(param)</code>	Is the value a valid port number, available for UDP listening.
<code>is_nonneg_int(param)</code>	Is the value a non-negative integer.
<code>is_bool(param)</code>	Is the value a boolean expression ("true", "false", "yes", "no", "1", "0").
<code>is_port(param)</code>	Is the value a valid port number (1-65536)
<code>is_pos_int(param)</code>	Is the value a positive integer.

You can also define your own validation rules using eval expressions that evaluate to true or false. Place the eval expression within a `validate()` function. See `eval` in the Splunk Search Reference for information on creating eval expressions.

For example, the following validation rules determine if the argument is in the form of a hyphen-separated Social Security number:

```
<arg name="ssn">
  <validation>
    validate(match('ssn', '^\d{3}-\d{2}-\d{4}$'), "SSN is not in valid format")
  </validation>
  . . .
</arg>
```

Another example defining a validation rule:



```

<arg name="bonus">
  <validation>
    validate(is_pos_int(bonus) AND bonus > 100, "Value must be a number greater than 100.")
  </validation>
  . . .
</arg>

```

### Single or multiple instances of a script

The default behavior for a script is to run in *one script instance per input stanza mode*. This results in multiple instances of the script, one for each input stanza. This default behavior is useful in multi-thread environments or in situations that require different security contexts or access to different databases.

In a single-threaded environment you might want to run in *single script instance mode*. For example, in a WMI environment you would run a single instance of a script so you can re-use connections.

You can override the default multiple instances of a script behavior by enabling *single script instance mode* in the introspection scheme:

```
<use_single_instance>true</use_single_instance>
```

### Introspection scheme and Splunk Manager pages

This section contains screen captures that illustrates how an introspection scheme affects the pages available from Settings.

Compare the screen captures here with the XML tags in the [Example scheme](#) listed above:

**Figure 1: Settings showing Modular Inputs with other data inputs**

Data inputs

Set up data inputs from files and directories, network ports, and scripted inputs. If you want to set up forwarding and receiving between two Splunk instances, go to [Forwarding and receiving](#).

Add data

Type	Inputs	Actions
<b>Files &amp; directories</b> Upload a file, index a local file, or monitor an entire directory.	4	<a href="#">Add new</a>
<b>TCP</b> Listen on a TCP port for incoming data, e.g. syslog.	0	<a href="#">Add new</a>
<b>UDP</b> Listen on a UDP port for incoming data, e.g. syslog.	0	<a href="#">Add new</a>
<b>Scripts</b> Run custom scripts to collect or generate more data.	0	<a href="#">Add new</a>
<b>Amazon S3</b> Get data from Amazon S3.	0	<a href="#">Add new</a>

Figure 2: Settings showing custom fields to add a modular input

Add new

Get data from Amazon S3.

Resource name \*  
An S3 resource name without the leading s3://. For example, for s3://bucket/file.txt specify bucket/file.txt.

Key ID \*  
Your Amazon key ID.

Secret key \*  
Your Amazon secret key.

☐ More settings

Cancel

Save

Read XML configuration from splunkd

A modular input script uses `stdin` to read `inputs.conf` configuration information from `splunkd`. The script parses the XML configuration information.

The XML format of the configuration information passed to the script depends on in which mode the script is running:

- single script instance per input stanza mode
- single script instance mode

**Note:** [Developer tools for modular inputs](#) in this manual shows how you can use the modular inputs utility to preview the configuration and the results returned by the script.

Configuration for single script instance per input stanza mode

In single script instance per input stanza mode, the XML configuration passed to the script looks something like this:

```
<input>
  <server_host>myHost</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>123102983109283019283</session_key>
  <checkpoint_dir>/opt/splunk/var/lib/splunk/modinputs</checkpoint_dir>
  <configuration>
    <stanza name="myScheme://aaa">
      <param name="param1">value1</param>
      <param name="param2">value2</param>
      <param name="disabled">0</param>
      <param name="index">default</param>
    </stanza>
  </configuration>
</input>
```

Tag	Description
<server_host>	The hostname for the Splunk Enterprise server.

Tag	Description
<code>&lt;server_uri&gt;</code>	The management port for the Splunk Enterprise server, identified by host, port, and protocol.
<code>&lt;session_key&gt;</code>	The session key for the session with splunkd. The session key can be used in any REST session with the local instance of splunkd.
<code>&lt;checkpoint_dir&gt;</code>	The directory used for a script to save checkpoints. This is where the input state from sources from which it is reading is tracked.
<code>&lt;configuration&gt;</code>	The child tags for <code>&lt;configuration&gt;</code> are based on the schema you define in the <code>inputs.conf.spec</code> file for your modular inputs. Splunk software reads all the configurations in the Splunk Enterprise installation and passes them to the script in <code>&lt;stanza&gt;</code> tags.

### **Configuration for single script instance mode**

The XML configuration information passed when running in single script instance mode varies slightly. When running in single script instance mode, all configuration stanzas have to be included because there is only one instance of the script running.

```
<input>
  <server_host>myHost</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>123102983109283019283</session_key>
  <checkpoint_dir>/opt/splunk/var/lib/splunk/modinputs</checkpoint_dir>
  <configuration>
    <stanza name="myScheme://aaa">
      <param name="param1">value1</param>
      <param name="param2">value2</param>
      <param name="disabled">0</param>
      <param name="index">default</param>
    </stanza>
    <stanza name="myScheme://bbb">
      <param name="param1">value11</param>
      <param name="param2">value22</param>
      <param name="disabled">0</param>
      <param name="index">default</param>
    </stanza>
  </configuration>
</input>
```

If you are running the modular input script in single script instance mode, and there are no configuration stanzas for your input scheme configured in `inputs.conf`, Splunk software passes in an empty configuration tag, as illustrated below. Your modular input script must be able to handle the empty configuration tag.

```
<input>
  <server_host>myHost</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>123102983109283019283</session_key>
  <checkpoint_dir>/opt/splunk/var/lib/splunk/modinputs</checkpoint_dir>
  <configuration/>
</input>
```

### **Example code reading XML configuration**

The following example shows how to read the XML configuration from splunkd. This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

```
# read XML configuration passed from splunkd
```

```

from builtins import str
def get_config():
    config = {}

    try:
        # read everything from stdin
        config_str = sys.stdin.read()

        # parse the config XML
        doc = xml.dom.minidom.parseString(config_str)
        root = doc.documentElement
        conf_node = root.getElementsByTagName("configuration")[0]
        if conf_node:
            logging.debug("XML: found configuration")
            stanza = conf_node.getElementsByTagName("stanza")[0]
            if stanza:
                stanza_name = stanza.getAttribute("name")
                if stanza_name:
                    logging.debug("XML: found stanza " + stanza_name)
                    config["name"] = stanza_name

                params = stanza.getElementsByTagName("param")
                for param in params:
                    param_name = param.getAttribute("name")
                    logging.debug("XML: found param '%s'" % param_name)
                    if param_name and param.firstChild and \
                        param.firstChild.nodeType == param.firstChild.TEXT_NODE:
                        data = param.firstChild.data
                        config[param_name] = data
                        logging.debug("XML: '%s' -> '%s'" % (param_name, data))

            checkpoint_node = root.getElementsByTagName("checkpoint_dir")[0]
            if checkpoint_node and checkpoint_node.firstChild and \
                checkpoint_node.firstChild.nodeType == checkpoint_node.firstChild.TEXT_NODE:
                config["checkpoint_dir"] = checkpoint_node.firstChild.data

            if not config:
                raise Exception("Invalid configuration received from Splunk.")

        # just some validation: make sure these keys are present (required)
        validate_conf(config, "name")
        validate_conf(config, "key_id")
        validate_conf(config, "secret_key")
        validate_conf(config, "checkpoint_dir")
    except Exception as e:
        raise Exception("Error getting Splunk configuration via STDIN: %s" % str(e))

    return config

```

## Enable, disable, and update modular input scripts

As with any other Splunk Enterprise app, you can enable, disable, or update the script that implements modular inputs. These actions produce the following behavior for modular inputs.

- **Disabling a modular input script**

When a modular input script is in the disabled state, the input is not initialized. The Settings pages do not reference the script. Splunk software ignores any inputs.conf files that reference the disabled modular input script.

If the modular input script is enabled, and then disabled while Splunk Enterprise is running, the script is stopped and unregistered. The endpoints for the script cannot be accessed and the Settings pages no longer reference

the script.

- Enabling a modular input script

If you enable a modular input script that was previously disabled, the script is registered with the Splunk platform. The endpoints for the script are accessible and the Settings pages for the script are available.

- Updating a modular input script

If you update a modular input script, then when it is enabled the previous version is disabled and the updated version is registered, updating the endpoints and Settings pages.

- Changes to other apps

If other apps are enabled, disabled, or updated, all active modular inputs reload. This is to ensure that updates to `inputs.conf` files properly reflect the modular inputs.

## Override default run behavior for modular input scripts

Adjust the `start_by_shell` parameter in `inputs.conf` to override default script running behavior for \*nix and Windows. This setting works similarly for scripted inputs and modular inputs. In most cases, the default setting does not need to be adjusted, but it can be set to `false` for scripts that do not need UNIX shell meta-character expansion.

The default settings for `start_by_shell` are:

- For \*nix: `true`. Scripts are passed to `/bin/sh -c`.
- For Windows: `false`. Scripts are started directly.

If the modular input runs in one-instance-per-stanza mode, override the default `start_by_shell` setting in the scheme default stanza. This setting is inherited by all of the scheme's input stanzas. You can also change the setting in any individual input stanza for more granular control.

If the modular input runs in single instance mode, override the default `start_by_shell` parameter setting in the scheme default stanza only. Other individual `start_by_shell` settings are ignored in this case.

## Set up logging

Well-behaved scripts send logging data to `splunkd.log`. This logging data is useful for tracking and troubleshooting.

### About logging

Any data you write to `stderr` is written to `splunkd.log`. You can specify a log level when writing to `stderr`. If unspecified, the log level defaults to `ERROR`. The following example shows how to write `INFO` and `ERROR` logging entries:

```
INFO Connecting to the endpoint
ERROR Unable to connect to the endpoint
```

Here are the recognized log levels from lowest to highest severity.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`
- `FATAL`

Log entries are written to `splunkd.log` based on the log level. By default, entries with a log level of `INFO` or higher are written to `splunkd.log`. To modify the default behavior, in Splunk Web navigate to **Settings > Server settings > Server logging**. Then navigate to the **ExecProcessor** log channel. Select **ExecProcessor** to make any changes.

Alternatively, you can navigate to the following file.

```
$SPLUNK_HOME/etc/log.cfg
```

In `log.cfg`, set the logging level for modular inputs by editing the log level in the following line.

```
category.ExecProcessor=INFO
```

For more information on logging, refer to What Splunk logs about itself in the Troubleshooting Manual.

**Note:** You must have Splunk Enterprise admin privileges to change logging behavior.

## Example: Setting up standard Splunk logging

The following snippet from a script shows how to set up standard Splunk logging.

### Standard Splunk logging snippets

```
. . .
import logging
. . .
# set up logging suitable for splunkd consumption
logging.root
logging.root.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(levelname)s %(message)s')
handler = logging.StreamHandler(stream=sys.stderr)
handler.setFormatter(formatter)
logging.root.addHandler(handler)
. . .
# add various logging statements
# for example:
#
# logging.info("URL %s already processed.  Skipping.")
#
#     if item_node:
#         logging.debug("XML: found item")
#
# etc.
```

## Set up external validation

In your modular input script, it is a good idea to validate the configuration of your input. Specify `<use_external_validation>true</use_external_validation>` in your introspection scheme to enable external validation.

If you provide an external validation routine and enable external validation the following occurs when a user creates or edits the configuration for a script:

1. Splunk software reads the configuration parameters from the user and creates an XML configuration of the parameters.

The XML configuration looks something like this:

```

<items>
  <server_host>myHost</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>123102983109283019283</session_key>
  <checkpoint_dir>/opt/splunk/var/lib/splunk/modinputs</checkpoint_dir>
  <item name="myScheme">
    <param name="param1">value1</param>
    <param_list name="param2">
      <value>value2</value>
      <value>value3</value>
      <value>value4</value>
    </param_list>
  </item>
</items>

```

**Notes:** The `<items>` element can only contain one `<item>`. (This is because you can only operate on one item at a time.) The XML stream itself must be encoded in UTF-8.

Refer to the [Read XML configuration from splunkd](#) section for a description of the XML configuration.

2. Splunk software invokes your script with the `--validate-arguments` option, passing in the XML configuration.

3. Your script validation routine determines if the configuration is valid.

- If the configuration is valid, your script exits with return status of zero.
- Otherwise the script exits with a non-zero status and a message indicating why configuration failed. Format the message in `<error>` tags so Splunk software can properly display the message in Splunk Web.

```

<error>
  <message>Access is denied.</message>
</error>

```

The following snippets shows how the [S3 example](#) validates data returned from the Amazon S3 service. The snippet at the end shows how to provide the `--validate-arguments` option when invoking the script. This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

### Validation snippets

```

...
from builtins import str
def get_validation_data():
    val_data = {}

    # read everything from stdin
    val_str = sys.stdin.read()

    # parse the validation XML
    doc = xml.dom.minidom.parseString(val_str)
    root = doc.documentElement

    logging.debug("XML: found items")
    item_node = root.getElementsByTagName("item")[0]
    if item_node:

```

```

logging.debug("XML: found item")

name = item_node.getAttribute("name")
val_data["stanza"] = name

params_node = item_node.getElementsByTagName("param")
for param in params_node:
    name = param.getAttribute("name")
    logging.debug("Found param %s" % name)
    if name and param.firstChild and \
        param.firstChild.nodeType == param.firstChild.TEXT_NODE:
        val_data[name] = param.firstChild.data

return val_data

# make sure that the amazon credentials are good
def validate_arguments():
    val_data = get_validation_data()

    try:
        url = "s3://" + val_data["stanza"]
        bucket, obj = read_from_s3_uri(url)
        conn = get_http_connection(val_data["key_id"], val_data["secret_key"], bucket, obj, method =
"HEAD")
        resp = conn.getresponse()
        log_response(resp)
        if resp.status != 200:
            raise Exception("Amazon returned HTTP status code %d (%s): %s" % (resp.status, resp.reason,
get_amazon_error(resp.read())))

    except Exception as e:
        print_error("Invalid configuration specified: %s" % str(e))
        sys.exit(1)
. . .
# Provide --validate-arguments arg on startup
if __name__ == '__main__':
    if len(sys.argv) > 1:
        if sys.argv[1] == "--scheme":
            do_scheme()
        elif sys.argv[1] == "--validate-arguments":
            validate_arguments()
        elif sys.argv[1] == "--test":
            test()
        else:
            usage()
    else:
        # just request data from S3
        run()

```

## Data checkpoints

When reading data for indexing you can set checkpoints to mark a source as having been read and indexed. You can persist any state information that is appropriate for your input. Typically, you store (check point) the progress of an input source so upon restart, the script knows where to resume reading data. This prevents you from reading and indexing the same data twice.

Splunk software provides a default location for storing checkpoints for modular inputs:

\$SPLUNK\_DB/modinputs/<input\_name>



For example, checkpoint data for the S3 example are stored here:

\$SPLUNK\_DB/modinputs/s3

## Enable checkpoints in your modular input script

The following example shows how to enable checkpoints in a script. This code sample is from the [Splunk S3 example](#).

### Create checkpoint files

In this snippet, you write a function to create the checkpoint file. The checkpoint file is an empty file with a unique name to identify it with the source. This example is encoding the url to an Amazon S3 source. This script has been made cross-compatible with Python 2 and Python 3 using python-future.

```
...
from builtins import range
def get_encoded_file_path(config, url):
    # encode the URL (simply to make the file name recognizable)
    name = ""
    for i in range(len(url)):
        if url[i].isalnum():
            name += url[i]
        else:
            name += "_"

    # MD5 the URL
    m = md5.new()
    m.update(url)
    name += "_" + m.hexdigest()

    return os.path.join(config["checkpoint_dir"], name)
...
# simply creates a checkpoint file indicating that the URL was checkpointed
def save_checkpoint(config, url):
    chk_file = get_encoded_file_path(config, url)
    # just create an empty file name
    logging.info("Checkpointing url=%s file=%s", url, chk_file)
    f = open(chk_file, "w")
    f.close()
...

```

### Test for checkpoint files

In this snippet, you have a function that tests if a checkpoint file exists. Call this function before reading from a source to make sure you don't read it twice.

```
...
# returns true if the checkpoint file exists
def load_checkpoint(config, url):
    chk_file = get_encoded_file_path(config, url)
    # try to open this file
    try:
        open(chk_file, "r").close()
    except:
        # assume that this means the checkpoint is not there
        return False
    return True
...

```

## Read a file and set a checkpoint

After reading a source, set a checkpoint. Here is how you checkpoint an Amazon S3 source.

```
. . .
if not load_checkpoint(config, url):
    # there is no checkpoint for this URL: process
    init_stream()
    request_one_object(url, key_id, secret_key, bucket, obj)
    fini_stream()
    save_checkpoint(config, url)
else:
    logging.info("URL %s already processed. Skipping.")
. . .
```

## Remove checkpoints

You can remove checkpoints by running the Splunk `clean` utility.

**Caution:** Be careful when removing checkpoints. Running the `clean` command removes your indexed data. For example, `clean all` removes ALL your indexed data.

For example, to remove checkpoints for a specific scheme:

```
splunk clean inputdata [<scheme>]
```

For example, to remove all checkpoints for the S3 modular input example, run the following command:

```
splunk clean inputdata s3
```

You can remove checkpoints for all modular inputs by running the command without the optional `<scheme>` argument. Or you could simply just use the `all` argument.

```
// Be careful with these commands! See CAUTION above.
```

```
splunk clean inputdata
splunk clean all
```

## Set up streaming

A modular input can stream data to a Splunk deployment as plain text or as XML data. In the schema for the modular input, use the `<streaming_mode>` tag to specify the streaming mode. Specify `simple` for plain text or `xml` for XML data.

For example, to specify XML data:

```
<streaming_mode>xml</streaming_mode>
```

### Simple streaming mode

Simple mode (plain text) is the default streaming mode and is similar to how Splunk software treats data that is streamed from scripted inputs. In simple mode, Splunk software treats the data much like it treats data read from a file. For more information on streaming from scripted inputs, refer to Scripted inputs overview in this manual.

In simple streaming mode, Splunk software supports all character sets described in Configure character set encoding

## XML streaming mode

With the Modular Inputs feature, new with Splunk 5.0, there is a new way to stream XML data to the Splunk platform. With this format for streaming XML you can:

- Clearly break events without the use of special markers.
- Easily forward data in a distributed environment by arbitrarily specifying done keys.
- Easily allow a single stream of data to specify source, sourcetype, host, and index.

The format of XML streaming differs, depending on which mode your script specifies:

- one script instance per input stanza mode
- single script instance mode

In XML streaming mode, the XML stream itself must be encoded in UTF-8.

## Default parameters when streaming events

The Splunk platform provides default values for the following parameters when streaming events. If Splunk software does not find a definition for these parameters in `inputs.conf` files, it uses the default values for these parameters.

source  
sourcetype  
host  
index

However, the default value varies, depending on whether you are using *one script instance per input stanza mode* or *single script instance mode*. The following table lists the default values for these parameters. The third column of the table lists the default values when using *traditional scripted inputs*.

Parameter	One script instance per input stanza	Single script instance	Traditional scripted inputs
<b>source</b>	scheme://  (for example, <i>myScheme://abc</i> )	scheme name  (for example, <i>myScheme</i> )	script://<path>  (<path> = <i>envvar-expanded path from inputs.conf</i> )
<b>sourcetype</b>	scheme name	scheme name	exec  (or if present, the layered value of the sourcetype)
<b>host</b>	Layered host for each stanza	Global default host from <code>inputs.conf</code>	Layered host from its stanza
<b>index</b>	Layered index for each stanza	Global default index from <code>inputs.conf</code>	Layered index from its stanza

## Specify the time of events in the input stream

If an input script knows the time of the event that it generates you can use the `<time>` tag to specify the time in the input stream. Specify the time using a UTC UNIX timestamp. Subseconds are supported (for example, `<time>1330717125.125</time>`).

**Note:** When writing modular input scripts, it is best to specify the time of an event with the tag. Splunk software does not read the timestamp from the body of the event (except in the case of unbroken events, described below). If a tag is not present, Splunk software attempts to use the time the data arrives from the input source as the time of the event.

When specifying the time of events, in `props.conf` set `SHOULD_LINEMERGE` to false. Refer to Configure event linebreaking in the *Getting Data In* manual for more information on setting this property.

Setting `SHOULD_LINEMERGE` to false does the following:

- Prevents the merging of events because of a missing timestamp.
- Does not override the value set with the `<time>` tag with a timestamp in the event.

The following example shows how to specify time events in the input stream:

```
<stream>
  <event stanza="my_config://aaa">
    <time>1330717125</time>
    <data>type=CCC</data>
  </event>
  <event stanza="my_config://bbb">
    <time>1330717125</time>
    <data>type=DDD</data>
  </event>
  . . .
</stream>
```

```
# Modify $SPLUNK_HOME/etc/apps/myapp/default/props.conf
```

```
[my_config]
SHOULD_LINEMERGE = false
```

## Streaming example (XML mode)

The streaming examples in XML mode in this section illustrate the differences between the following:

- one script instance per input stanza mode
- single script instance mode

The examples also show how you can override the default values for the following parameters:

- source
- sourcetype
- host
- index

**Note:** For these examples, the introspection scheme enables XML streaming mode, as described in [Define a scheme for introspection](#).

### ***One script instance per input stanza mode***

This example shows some example XML that a script can stream to splunkd for indexing, using one script instance per input stanza mode. In this mode, there is a separate instance of the script for each input stanza in `inputs.conf` configuration files.

```
<stream>
  <event>
    <time>1370031029</time>
    <data>event_status="(0)The operation completed successfully."</data>
  </event>
  <event>
    <time>1370031031</time>
    <data>event_status="(0)The operation completed successfully."</data>
  </event>
</stream>
```

In this example, the tags clearly delineate the events. This effectively line-breaks the events without any line-breaking configuration.

The values for source, sourcetype, host, and index are the default values, as described in [Default parameters when streaming events](#). You can override the default values by including the new values in the event. The following example specifies custom values for source and index:

```
<stream>
  <event>
    <time>1370031035</time>
    <data> event_status="(0)The operation completed successfully."</data>
    <source>my_source</source>
    <index>test1</index>
  </event>
</stream>
```

**Note:** Subsequent events can specify new values for the source and index parameters, or simply use the default values.

### ***Single script instance mode***

This example shows some example XML that a script can stream to splunkd for indexing, using single script instance mode. In this mode, there is only a single instance of the script.

**Note:** Because you are using a single instance of the script, use the stanza attribute to the `<event>` tag to specify the stanza for each event. Specifying the stanza attribute is not needed when streaming in one script instance per input stanza mode.

```
<stream>
  <event stanza="my_config://aaa">
    <time>1370031041</time>
    <data> event_status="(0)The operation completed successfully."</data>
    <host>my_host</host>
  </event>
```

```
</stream>
```

In this example, the value of stanza should be an existing stanza name from `inputs.conf` that the event belongs to. If the stanza name is not present (or refers to a non-existent stanza name in the conf file) then Splunk software automatically sets the parameters for source, sourcetype, host, and index.

This example overrides the default value for the host parameter.

## Stream unbroken events in XML

The XML streaming examples in the previous sections use the `<event>` tag to delineate, or break, separate events. However, often when you stream data to the Splunk platform, you do not want to break events, and instead let Splunk software interpret the events. You typically send unbroken data in chunks and let Splunk software apply line breaking rules.

You may want to stream unbroken events either because you are streaming a known format to the Splunk platform, or you may not know the format of the data and you want Splunk software to interpret it. The [S3 example](#) in this document streams unbroken events in XML mode.

### *Use the `<time>` tag when possible*

When streaming unbroken events, Splunk software attempts to read timestamps from the body of the events, and break the event based on the timestamps. However, if known, the tag should be provided for unbroken events. When the unbroken segments are merged, the value from the first tag is used. However, it may be overridden by any timestamp extraction rules for the sourcetype.

### *Use the `<done>` tag with unbroken events*

Use the `<done>` tag to denote an end of a stream with unbroken events. The `<done>` tag tells Splunk software to flush the data from its buffer rather than wait for more data before processing it. For example, Splunk software may buffer data that it has read, waiting for a newline character before processing the data. This prevents the data from being indexed until the newline character is read. If you want the data indexed without the newline character, then send the `<done>` tag.

Specify the `unbroken` attribute to the `<event>` tag. Then after you have reached the end of the data you are sending in chunks, send the `<done/>` tag as indicated in the following example.

```
<stream>
  <event unbroken="1">
    <data>09/08/2009 14:01:59.0398 part of the event ...</data>
  </event>
  <event unbroken="1">
    <data>final part of the event</data>
    <done/>
  </event>
  <event unbroken="1">
    <data>second event</data>
    <done/>
  </event>
</stream>
```

When sending unbroken events:

- You can specify source, sourcetype, host, index, and stanza specifications just as you would when sending broken events.

- The script is responsible for sending a `<done/>` tag. This is important for forwarders because they can't switch a stream until they see a `<done/>` tag.
- When the data goes through the time extraction process, if a subset of the event is identified as a timestamp, that time becomes the event's time, and the timestamp is used for event aggregation. Refer to [Configure event linebreaking](#) for more information.

## Modular inputs configuration

This topic describes several ways to define configuration for modular inputs. It includes the following:

- How to create and edit the `inputs.conf.spec` file for modular inputs.
- Configuration layering for modular inputs
- Specifying permissions to access modular input apps

### Create a modular input spec file

Specific locations are required for all spec files. For modular inputs, the spec file is located in a `README` directory of the app implementing the modular input.

```
$SPLUNK_HOME/etc/apps/<myapp>/README/inputs.conf.spec
```

The location of script referenced in the spec file is here:

```
$SPLUNK_HOME/etc/apps/<myapp>/bin/<myscript>
```

#### **Structure of a spec file**

Splunk Enterprise provides numerous spec files that it uses to configure and access a Splunk Enterprise server. These default spec files are heavily commented and include examples on how to configure Splunk Enterprise.

However, the structure of a spec file is quite basic, it only requires the following elements:

- stanza header (one or more)
- param values (one or more for each stanza)

The following shows a minimal `inputs.conf.spec` file. In this file, the values for the parameters are not present. These are not required. If present, Splunk Enterprise ignores them. Additionally, the `<name>` element in the stanza header is ignored.

#### **Sample inputs.conf.spec file**

```
[myscript://<name>]
param1 =
```

#### **Writing valid spec files**

Here are some things to keep in mind when writing spec files:

- The `inputs.conf.spec` spec file must be at the following location:

```
$SPLUNK_HOME/etc/apps/<app_name>/README/
```

- The following regex defines valid identifiers for the scheme name (the name before the ://) and for parameters:

```
[0-9a-zA-Z][0-9a-zA-Z_-]*
```

- Avoid name collision with built-in scheme names. Do not use any of the following as scheme names for your modular inputs:

```
batch
fifo
monitor
script
splunktcp
tcp
udp
```

- Some parameters are always implicitly defined. Specifying any of the following parameters for your modular inputs has no effect. However, you could specify these to help clarify the usage:

```
source
sourcetype
host
index
disabled
interval
persistentQueue
persistentQueueSize
queueSize
```

- Modular inputs can only be defined once. Subsequent definitions (a new scheme stanza) and their parameters are ignored.
- A scheme must define at least one parameter. Duplicate parameters are ignored.
- The stanza definition and their parameters must start at the beginning of the line.

### ***Spec file example***

Here is the spec file for the Amazon S3 example.

### **S3 inputs.conf.spec file**

```
[s3://<name>]
key_id = <value>
* This is Amazon key ID.
```

```
secret_key = <value>
* This is the secret key.
```

### **Configuration layering for modular inputs**

As described in Configuration file precedence in the Admin manual, Splunk Enterprise uses configuration layering across `inputs.conf` files in your system. Configuration for modular inputs contrasts with how configuration generally works. Typically a configuration stanza only inherits from the global default configuration.



For modular inputs configuration, each modular input scheme gets a separate default stanza in `inputs.conf`. After Splunk Enterprise layers the configurations, the configuration stanza for a modular input (`myScheme://aaa`) inherits values from the scheme default configuration. A modular input can inherit the values for `index` and `host` from the `default` stanza, but the scheme default configuration can override these values.

For example, consider the following `inputs.conf` files in a system:

### **Global default**

**.../etc/system/local/inputs.conf**

```
[default]
. . .
index = default
host = myHost
```

### **Scheme default**

**.../etc/apps/myApp/default/inputs.conf**

```
[myScheme]
host = myOtherHost
param1 = p1
```

### **Configuration stanza**

**.../etc/apps/search/local/inputs.conf**

```
[myScheme://aaa]
param2 = p2
```

Here is how layered configuration is built:

1. Apply the values for `index` and `host` from the global default.  
In a typical installation the values for `index` and `host` from the global default configuration apply to all inputs. Other values in the global default configuration do not apply to modular inputs.
2. Apply values from scheme default, overriding any values previously set.
3. Apply values from configuration stanza, overriding any values previously set.

The layered outcome of the above configuration example is:

### **Layered configuration example**

```
[myScheme://aaa]
index = default      #from Global default
host = myHost       #from Global default, overridden by Scheme default
host = myOtherHost   #from Scheme default
param1 = p1          #from Scheme default
param2 = p2          #from Configuration stanza
```

## Interval parameter

Use the interval parameter to schedule and monitor scripts. The interval parameter specifies how long a script waits before it restarts.

The interval parameter is useful for a script that performs a task periodically. The script performs a specific task and then exits. The interval parameter specifies when the script restarts to perform the task again.

The interval parameter is also useful to ensure that a script restarts, even if a previous instance of the script exits unexpectedly.

Entering an empty value for interval results in a script only being executed on start and/or endpoint reload (on edit).

### *single script instance per input stanza mode*

For single script instance per input stanza mode, each stanza can specify its own interval parameter.

### *single script instance mode*

For single script instance mode, Splunk Enterprise reads the interval setting from the scheme default stanza only. If interval is set under a specific input stanza, that value is ignored.

For single script instance mode, interval cannot be an endpoint argument, even if it is specified in inputs.conf.spec. You cannot modify the interval value for single script instance mode using the endpoint.

## Persistent queues

You can configure persistent queues with modular inputs. You can use persistent queues with modular inputs much as you do with TCP, UDP, FIFO, and scripted inputs, as described in Use persistent queues to help prevent data loss.

You configure persistent queues for modular inputs much as you do with other inputs. There are differences depending on the type of modular input.

### *single script instance per input stanza mode*

In this mode, a script is spawned for each inputs stanza. Because each script produces its own stream, it can have its own persistent queue. The correct way to configure a persistent queue is to put the persistent queue parameters under each inputs stanza:

```
[foobar://aaa]
param1 = 1234
param2 = qwerty
queueSize = 50KB
persistentQueueSize = 100MB
```

Another way to configure a persistent queue is to put queueSize and persistentQueueSize under the scheme default stanza (in this example, [foobar]). All input stanzas inherit these params and result in the creation of a separate persistent queue for each input stanza.

### ***single script instance mode***

In this mode, there is only one stream of data that services all inputs stanzas for that modular input. The only valid way to configure the persistent queue is to put the settings under the scheme default stanza. Placing it under a specific input stanza has no effect.

```
[foobar]
queueSize = 50KB
persistentQueueSize = 100MB
```

### ***Persistent queue location***

Persistent queue files are in the same directory location as scripted inputs:

`$SPLUNK_HOME/var/run/splunk/exec/<encoded path>`

`<encoded path>` derives from the inputs stanza (for single script instance per input stanza mode) or the scheme name (for single script instance mode).

## **Specify permissions for modular input scripts**

Read permission for modular input scripts is controlled by the `list_inputs` capability. This capability also controls reading of other input endpoints.

By default, the `admin_all_objects` capability controls create and edit permissions for modular inputs. However, you have the option to create a capability that customizes edit and create permissions for any specific modular input scheme. If the custom capability for a modular input is present, the custom capability is applied rather than the default `admin_all_objects` capability.

The custom capability for modular inputs takes the following form:

```
edit_modinput_myscheme
```

After creating the capability for a modular input, enable it for one or more user roles.

**Caution:** Make sure you assign one or more roles for the capability `edit_modinput_myscheme`, otherwise no one can create or edit modular inputs for that scheme.

To create a custom capability and assign roles edit the `authorize.conf` configuration file. For example, to create a custom create and edit capability for the MyScheme modular input, and then enable it for the admin and power roles, do the following:

**`$SPLUNK_HOME/etc/apps/<app_name>/default/authorize.conf`**

```
[capability::edit_modinput_MyScheme]
```

```
[role_admin]
edit_modinput_MyScheme = enabled
```

```
[role_power]
edit_modinput_MyScheme = enabled
```

For more information on roles and capabilities, refer to:

- About defining roles and capabilities in the Securing Splunk Enterprise manual
- authorize.conf spec file in the Configuration File Reference

## Create a custom user interface

You can create a custom Manager page for modular inputs that gives you more flexibility in the content displayed. The custom page overrides the Splunk Manager page your modular input script defines during introspection. See [Define a scheme for introspection](#) for details on how introspection defines a Manager page.

Here are the steps for creating a custom Manager page:

1. Create a `manager.xml` file that defines the user interface.
2. Set sharing for your modular input script so others can access the manager pages.
3. Restart Splunk instance.

**Caution:** Creating a custom user interface for modular scripts is an advanced topic. You should have familiarity with the Splunk Enterprise framework and be comfortable editing Splunk Enterprise system files. Modifying existing manager pages or creating new ones affects how users interact with the Splunk server through Splunk Web.

### Manager XML files

Splunk Enterprise uses manager XML files in the following `manager` directory to define the contents of pages in Splunk Manager.

```
$SPLUNK_HOME/etc/apps/<App>/default/data/ui/manager/<ManagerFile>.xml
```

The manager pages provides a user interface to create, update, and list Splunk Enterprise resources. For modular inputs, you can create a custom interface for the inputs defined in your script.

The names of the files in the `manager` directory are not important. Splunk Enterprise searches all files in the directory when building the manager pages for Splunk Web.

In your Splunk Enterprise installation, you can access the Manager page implementations for the default Search app:

```
$SPLUNK_HOME/etc/apps/search/default/data/ui/manager/*.xml
```

### *Manager pages for modular input scripts*

To define a custom Manager page for a modular input script, place a manager XML file at the following location:

```
$SPLUNK_HOME/etc/apps/<myApp>/default/data/ui/manager/<ManagerFile>.xml
```

Name the manager file anything you like. As described above, Splunk Enterprise checks the contents of the `manager` directory, searching for XML files in the correct format. Use the name of the modular input script in the name of the manager file.

You can study the contents of existing manager files for data inputs to get some ideas for implementing your own. For example, compare:

```
$SPLUNK_HOME/etc/apps/search/default/data/ui/manager/data_inputs_script.xml
```

to the page:

Manager > Data inputs > Script > Add new

**Caution:** Be careful not to make any changes to existing manager files.

### Create and edit manager XML files

When creating a manager XML file, make sure you accurately specify references to your modular inputs and create your widgets correctly. Here are some tips to get you started:

- In the top-level tag, `<endpoint name="...">`, make sure `name` correctly points to the path to the modular input endpoint.
- For the `<breadcrumb>` tag, make sure you specify the following:

```
<parent hidecurrent="False">datainputstats</parent>
```

Also specify the name of your script for the `<name>` tag.

- For `<element name="...">` tags, `name` refers to a field defined for your modular inputs in `inputs.conf`.
- For individual elements representing widgets, study the modular input examples, and also the data-inputs manager files for the Splunk Search app.

### Manager XML file tags

The following table describes the tags available to create a manager XML file. Not all tags are detailed. For examples of available tags, see the manager files for the default Search app, as described previously.

Tag	Description
<b><code>&lt;endpoint name=path to endpoint&gt;</code></b>	<p>Top-level tag. <code>&lt;header&gt;</code>, <code>&lt;breadcrumb&gt;</code>, and <code>&lt;elements&gt;</code> are child tags to <code>&lt;endpoint&gt;</code>.</p> <pre>name = data/inputs/&lt;scriptName&gt;</pre> <p>The <code>name</code> attribute provides the path to the Splunk endpoint for your script. <code>&lt;scriptName&gt;</code> is the name of your modular input script.</p> <p>The endpoint path to modular input endpoints are always in the form listed above for <code>name</code>.</p>
<b><code>&lt;header&gt;</code></b>	<p>Required. Child tag to <code>&lt;endpoint&gt;</code>.</p> <p>The title Splunk Web displays to access the manager page for your modular input.</p>
<b><code>&lt;breadcrumb&gt;</code></b>	<p>Recommended. Child tag to <code>&lt;endpoint&gt;</code></p> <p>Use this tag to specify breadcrumb links to your manager page.</p> <p>For modular inputs, you typically specify the following:</p>

Tag	Description
	<pre>&lt;parent hidecurrent="False"&gt;datainputstats&lt;/parent&gt; &lt;name&gt;Script name&lt;/name&gt;</pre>
<b>&lt;elements&gt;</b>	<p>Required child tag to &lt;endpoint&gt;. Optional child tag to &lt;element&gt;.</p> <p>Tag containing the &lt;element&gt; tags. You can nest &lt;elements&gt; within an &lt;element&gt; tag.</p>
<b>&lt;element&gt;</b>	<p>Required. Child tag to &lt;elements&gt;.</p> <p>Defines the user interface elements for the manager page.</p> <p>&lt;element&gt; can take the following attributes:</p> <p>name: For modular inputs, corresponds to a field name listed in inputs.conf. Can also take a value beginning with "spl-ctrl_." In this case, the element is not bound to a field name, but instead serves a cosmetic purpose.</p> <p>type: Defines the widget to display. See below for widgets available.</p> <p>label: Text field describing the widget.</p> <p>&lt;element&gt; can take the following child tags:</p> <pre>&lt;view&gt; &lt;elements&gt;</pre> <p>The &lt;view&gt; tags has additional child tags that define the widgets and accompanying text in the manager page. See the following section, <a href="#">The element tag</a>, for details on child tags to &lt;element&gt; and &lt;view&gt;.</p>

#### The Element tag

An <element> defines a widget to use in a manager page. You can nest widgets inside other widgets. The widgets available are in the following `widgets` directory.

`$SPLUNK_HOME/share/splunk/search_mrsparkle/templates/admin/widgets/`

Specify a name and a type for the <element>.

```
<element type="checkbox" name="my_checkbox"> [...]
```

Add `<view name="...">` tags to specify the views in which the <element> should be visible.

- create: This view creates a new instance of the element.
- edit: This view edits an existing instance of the element
- list: The element appears in views that list all elements.

For examples, see the data inputs manager files available in the following `search` app directory.

`$SPLUNK_HOME/etc/apps/search/default/data/ui/manager`

## Manager page example

Here is an example manager XML file for S3.

**\$SPLUNK\_HOME/etc/apps/s3/default/data/ui/manager/s3.xml**

```
<endpoint name="data/inputs/s3">
  <header>Amazon S3</header>
  <breadcrumb>
    <parent hidecurrent="False">datainputstats</parent>
    <name>S3</name>
  </breadcrumb>
  <elements>
    <element name="sourceFields" type="fieldset">
      <key name="legend">Source</key>
      <view name="list"/>
      <view name="edit"/>
      <view name="create"/>
      <elements>
        <element name="name" label="Resource name">
          <view name="list"/>
          <view name="create"/>
        </element>
        <element name="key_id" type="password" label="Key ID">
          <view name="edit"/>
          <view name="create"/>
          <key name="exampleText">Your Amazon key ID. OZRAA</key>
        </element>
        <element name="secret_key" type="password" label="Secret key">
          <view name="edit"/>
          <view name="create"/>
          <key name="exampleText">Your Amazon secret key.</key>
        </element>
      </elements>
    </element>

    <element name="sourcetypeFields" type="fieldset">
      <view name="list"/>
      <view name="edit"/>
      <view name="create"/>
      <elements>
        <element name="spl-ctrl_sourcetypeSelect" type="select" label="Set the source type">
          <onChange>
            <key name="auto">NONE</key>
            <key name="_action">showonly</key>
            <group_set>
              <group name="sourcetype"/>
              <group name="spl-ctrl_from_list"/>
            </group_set>
            <key name="sourcetype">sourcetype</key>
            <key name="spl-ctrl_from_list">spl-ctrl_from_list</key>
          </onChange>
          <options>
            <opt value="auto" label="Automatic"/>
            <opt value="sourcetype" label="Manual"/>
            <opt value="spl-ctrl_from_list" label="From list"/>
          </options>
          <view name="edit"/>
          <view name="create"/>
        </element>
      </elements>
    </element>
  </elements>
</endpoint>
```

```

<key name="exampleText">When this is set to automatic, Splunk classifies and
    assigns the sourcetype automatically, and gives unknown sourcetypes
    placeholder names.</key>
<key name="processValueEdit">[[ e for e in ['sourcetype']
    if form_defaults.get(e) ][0]]</key>
<key name="processValueAdd">[[ e for e in ['sourcetype']
    if form_defaults.get(e) ][0]]</key>
</element>
<element name="sourcetype" type="textfield" label="Source type">
    <view name="list"/>
    <view name="edit"/>
    <view name="create"/>
    <key name="processValueList">_('Automatic') if (value==None or value=='') else value</key>
    <key name="submitValueAdd">value if
        form_data.get('spl-ctrl_sourcetypeSelect')== 'sourcetype'
        else (form_data.get('spl-ctrl_from_list')
            if form_data.get('spl-ctrl_sourcetypeSelect')== 'spl-ctrl_from_list'
            else '')</key>
    <key name="submitValueEdit">value if
        form_data.get('spl-ctrl_sourcetypeSelect')== 'sourcetype'
        else (form_data.get('spl-ctrl_from_list')
            if form_data.get('spl-ctrl_sourcetypeSelect')== 'spl-ctrl_from_list'
            else '')</key>
    <key name="labelList">Source type</key>
</element>
<element name="spl-ctrl_from_list" type="select" label="Select source type from list">
    <view name="edit"/>
    <view name="create"/>
    <key name="exampleText">Splunk classifies all common data types automatically,
        but if you're looking for something specific, you can find more source types
        in the <![CDATA[<a href=".../apps/remote">Splunk Apps apps browser</a>]]>
        or online at <![CDATA[<a href="http://splunkbase.splunk.com/"
        target="_blank">http://splunkbase.splunk.com</a>]]>.</key>
    <key name="requiredIfVisible" />
    <key name="dynamicOptions" type="dict">
        <key name="prefixOptions" type="list">
            <item type="list">
                <item></item>
                <item>Choose...</item>
            </item>
        </key>
        <key name="keyName">title</key>
        <key name="keyValue">title</key>
        <key name="splunkSource">/saved/sourcetypes</key>
        <key name="splunkSourceParams" type="dict">
            <key name="count">-1</key>
            <key name="search">'pulldown_type=true'</key>
        </key>
    </key>
</element>
</elements>
<key name="legend">Source type</key>
<key name="helpText">Tell Splunk what kind of data this is so you can group it with
    other data of the same type when you search. Splunk does this
    automatically, but you can specify what you want if Splunk gets it
    wrong.</key>
</element>

<element name="spl-ctrl_EnableAdvanced" type="checkbox"
    label="More settings" class="spl-mgr-advanced-switch">
    <view name="edit"/>
    <view name="create"/>

```



```

<onChange>
  <key name="_action">showonly</key>
  <key name="0">NONE</key>
  <key name="1">ALL</key>
  <group_set>
    <group name="advanced"/>
  </group_set>
</onChange>
</element>
<element name="advanced" type="fieldset" class="spl-mgr-advanced-options">
  <view name="edit"/>
  <view name="create"/>
  <elements>
    <element name="hostFields" type="fieldset">
      <key name="legend">Host</key>
      <view name="list"/>
      <view name="edit"/>
      <view name="create"/>
      <elements>
        <element name="host" type="textfield" label="Host field value">
          <view name="edit"/>
          <view name="create"/>
        </element>
      </elements>
    </element>
    <element name="indexField" type="fieldset">
      <key name="legend">Index</key>
      <key name="helpText">Set the destination index for this source.</key>
      <view name="list"/>
      <view name="edit"/>
      <view name="create"/>
      <elements>
        <element name="index" type="select" label="Index">
          <view name="list"/>
          <view name="edit"/>
          <view name="create"/>
          <key name="dynamicOptions" type="dict">
            <key name="keyName">title</key>
            <key name="keyValue">title</key>
            <key name="splunkSource">/data/indexes</key>
            <key name="splunkSourceParams" type="dict">
              <key name="search">'isInternal=false disabled=false'</key>
              <key name="count">-1</key>
            </key>
          </key>
        </element>
      </elements>
    </element>
  </elements>
</element>
<element name="eai:acl.app" label="App">
  <view name="list"/>
  <key name="processValueList">entity['eai:acl']['app'] or ""</key>
</element>

</elements>
</endpoint>

```

## Set sharing for your modular input script

You need to share your modular inputs script before Splunk Manager pages are visible to other users. Typically, you share your script so all users can access it.

To share your modular input script, create the following `default.meta` file:

`$SPLUNK_HOME/etc/apps/<myApp>/metadata/default.meta`

```
[]  
export = system
```

## Restart Your Splunk Instance

After creating or updating manager pages for modular inputs, and also updating sharing for your modular inputs script, restart your Splunk instance for the changes to take effect.

## Developer tools for modular inputs

### REST API access

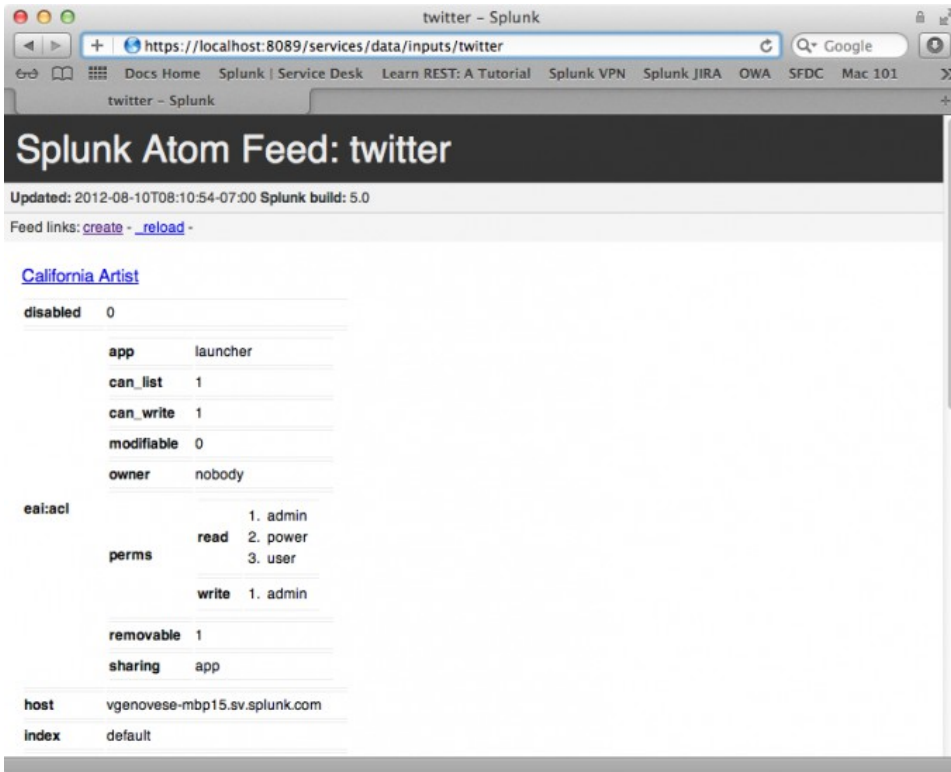
The Splunk platform provides REST endpoints to access modular inputs installed on a Splunk server. You can access the REST endpoint using the manager port of Splunk Web.

```
https://localhost:8089/services/data/modular-inputs  
https://localhost:8089/services/data/modular-inputs/{name}
```

Details of the endpoints are available from the REST API Reference Manual, which the following list links to.

- `data/modular-inputs`  
Lists all modular inputs
- `data/modular-inputs/{name}`  
Provides details on a specific modular endpoint.

The following screen capture shows how Splunk Web displays the return values from the `data/modular-inputs/twitter` endpoint, which is the Twitter example application.



## Modular inputs configuration utility

When developing a modular input script, it is useful to run the script in isolation, outside of the context of the Splunk server. You can do this using the Splunk utility, `print-modinput-config`. With this utility you can:

- View the configuration XML generated from a stanza in `inputs.conf`.
- View verbose debugging information.
- Pipe the configuration into an instance of the script to preview the output

### *Print modular inputs configurations*

Use the Splunk utility, `print-modinput-config` to print the XML configuration for a modular input. Here is how you call the command for a script named `myscript.py` with the specified stanza in `inputs.conf`.

```
splunk cmd splunkd print-modinput-config myscheme mystanza
```

**Note:** You can run the script with the `--debug` parameter to view verbose debugging information generated by your script.

For example, suppose you have a modular input script, `twitter.py` and the following stanza for the script in your `inputs.conf` file:

### inputs.conf

```
[twitter://SplunkTwitter]
password = pass
```

```
username = splunker
```

Run the utility to view the configuration for this input:

**splunk cmd splunkd print-modinput-config twitter twitter://SplunkTwitter**

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <server_host>vgenovese-mbp15.sv.splunk.com</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>035ec937131efaa14116dffcd3f3fe6</session_key>
  <checkpoint_dir>/Applications/splunk/var/lib/splunk/modinputs/twitter</checkpoint_dir>
  <configuration>
    <stanza name="twitter://SplunkTwitter">
      <param name="host">vgenovese-mbp15.sv.splunk.com</param>
      <param name="index">default</param>
      <param name="password">pass</param>
      <param name="username">splunker</param>
    </stanza>
  </configuration>
</input>
```

This is the configuration information that Splunk software passes to your script when the script is invoked. You can then pipe this configuration information back into your script. This simulates running the modular input script under splunkd and can be used to debug the event stream output..

**splunk cmd splunkd print-modinput-config twitter twitter://SplunkTwitter \**  
**| splunk cmd python \$SPLUNK\_HOME/etc/apps/twitter/bin/twitter.py**

```
DEBUG XML: found configuration
DEBUG XML: found stanza twitter://SplunkTwitter
. . .
{"user": {"default_profile": false, "id_str": "22268633", "statuses_count": 27 \
03, "location": "Ja\u00e9n, Andalucia, Spain", "profile_background_image_url": \
"http://a0.twimg.com/profile_background_images/451794773/afhbanner.jpg", "fol \
lowers_count": 933, "id": 22268633, "contributors_enabled": false, "follow_req \
uest_sent": null, "profile_background_tile": true, "is_translator": false, "pr \
ofile_sidebar_border_color": "99CC5C", "profile_image_url_https": "https://si0 \
. . .
```

### ***Debug mode for printing modular inputs configuration***

You can run the modular inputs configuration script in debug mode to get verbose debugging information for all modular inputs configurations on your system.

For example, specify the following to view debug information for the Splunk Twitter example. Debug prints information, not just for the Twitter example, but for additional modular inputs on your system. The results for S3 modular inputs have been elided for brevity.

**splunk cmd splunkd print-modinput-config --debug twitter twitter://SplunkTwitter**

```
Found scheme="s3".
Locating script for scheme="s3"...
. . .
Introspection setup completed for scheme "s3".
=====
Found scheme="twitter".
Locating script for scheme="twitter"...
```

```

No regular file="/Applications/splunk/etc/apps/twitter/darwin_x86_64/bin/twitter.sh".
No regular file="/Applications/splunk/etc/apps/twitter/darwin_x86_64/bin/twitter.py".
No regular file="/Applications/splunk/etc/apps/twitter/darwin_x86_64/bin/twitter".
No script found in dir="/Applications/splunk/etc/apps/twitter/darwin_x86_64/bin"
No regular file="/Applications/splunk/etc/apps/twitter/bin/twitter.sh".
Found script "/Applications/splunk/etc/apps/twitter/bin/twitter.py" to handle scheme "twitter".
XML scheme path "/scheme/title": "title" -> "Twitter"
XML scheme path "/scheme/description": "description" -> "Get data from Twitter."
XML scheme path "/scheme/use_external_validation": "use_external_validation" -> "true"
XML scheme path "/scheme/streaming_mode": "streaming_mode" -> "simple"
XML arg path "/scheme/endpoint/args/arg": "name" -> "name"
XML arg path "/scheme/endpoint/args/arg/title": "title" -> "Twitter feed name"
XML arg path "/scheme/endpoint/args/arg/description": "description" -> "Name of the current feed using the
user credentials supplied."
XML arg path "/scheme/endpoint/args/arg": "name" -> "username"
XML arg path "/scheme/endpoint/args/arg/title": "title" -> "Twitter ID/Handle"
XML arg path "/scheme/endpoint/args/arg/description": "description" -> "Your Twitter ID."
XML arg path "/scheme/endpoint/args/arg": "name" -> "password"
XML arg path "/scheme/endpoint/args/arg/title": "title" -> "Password"
XML arg path "/scheme/endpoint/args/arg/description": "description" -> "Your twitter password"
Setting up values from introspection for scheme "twitter".
Setting "title" to "Twitter".
Setting "description" to "Get data from Twitter.".
Setting "use_external_validation" to true.
Setting "streaming_mode" to "simple".
Endpoint argument settings for "name":
Setting "title" to "Twitter feed name".
Setting "description" to "Name of the current feed using the user credentials supplied.".
Endpoint argument settings for "password":
Setting "title" to "Password".
Setting "description" to "Your twitter password".
Endpoint argument settings for "username":
Setting "title" to "Twitter ID/Handle".
Setting "description" to "Your Twitter ID.".
Introspection setup completed for scheme "twitter".
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <server_host>vgenovese-mbp15.local</server_host>
  <server_uri>https://127.0.0.1:8089</server_uri>
  <session_key>8586adf254cce215630e8c022d1f3c7c</session_key>
  <checkpoint_dir>/Applications/splunk/var/lib/splunk/modinputs/twitter</checkpoint_dir>
  <configuration>
    <stanza name="twitter://SplunkTwitter">
      <param name="host">vgenovese-mbp15.sv.splunk.com</param>
      <param name="index">default</param>
      <param name="password">pass</param>
      <param name="username">splunker</param>
    </stanza>
  </configuration>
</input>

```

## Input status endpoint

The input status endpoint is useful when troubleshooting modular inputs. It can help you determine issues such as the following:

- Is a modular input script running?
- Why is there no searchable data?
- How much data did the script stream?

The input status is available from the following management endpoint:

https://localhost:8089/services/admin/inputstatus

**Note:** 8089 is the default management port. Your management port may be different.

From the management endpoint for input status, you can find a link to `ModularInputs:modular input` command that lists all modular inputs, their location in the system, the number of bytes indexed, and their status.

For example, here is the input status for the Twitter modular input example:

## Splunk Atom Feed: inputstatus

Updated: 2012-07-06T11:45:50-07:00 Splunk build: 5.0

[ModularInputs:modular input commands](#)

eai:aci	app		
	can_list	1	
	can_write	1	
	modifiable	0	
	owner	system	
	perms	read 1, *	
		write	
	removable	0	
	sharing	system	
	optionalFields		
eai:attributes	requiredFields		
	wildcardFields		
inputs	/Applications/splunk/etc/apps/twitter/bin/twitter.py (twitter://California Artist)	time opened	2012-07-06T11:45:15-0700
		total bytes	4145152
	/Applications/splunk/etc/apps/twitter/bin/twitter.py (twitter://SplunkTwitter)	time opened	2012-07-06T11:42:54-0700
		total bytes	20578304

[list](#) -  
2012-07-06T11:45:50-07:00 | [system](#)

The input status endpoint only includes actual data. In the case of XML streaming, it only includes the number of bytes within the `<data>` tags. If a script has started and then exited for whatever reason, the exit status description contains a human-readable string that explains why the script exited. For example it may say "exited with code 0."

## Track a modular input script

If your script provides any type of logging to stderr (for example a logger output, or a python stack trace printed to stderr by the interpreter), these contents are written to `splunkd.log`, as described in the section [Set up logging](#).

You can search the log file to retrieve the logging data. The following example searches for the output from any script spawned by the modinputs framework. This includes any messages from the ExecProcessor system component, which is responsible for running and managing the scripts. You can modify this search according to your specific needs.

```
index=_internal source=*splunkd.log* (component=ModularInputs stderr) OR component=ExecProcessor
```

## Modular inputs examples

These examples use Python for the scripting language. However, you can use various other scripting languages to implement modular inputs.

**Note:** Splunk Universal Forwarder, unlike other Splunk instances, does not provide a Python interpreter. In this case, to run these examples you may need to install Python on the server if one is not already available.

### Twitter example

The Twitter example streams JSON data from a Twitter source to the Splunk platform for indexing.

**Note:** The example uses Tweepy, a Python library, to access the Twitter source. Tweepy libraries must be available to the Splunk Twitter example script, `twitter.py`. To run the example, download and install Tweepy.

#### *Twitter example script*

Place the `twitter.py` script in the following location in your Splunk installation:

```
$SPLUNK_HOME/etc/apps/twitter/bin/twitter.py
```

Refer to [Scripts for modular inputs](#) for analysis of specific parts of the script. This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

#### twitter.py

```
from __future__ import print_function
from future import standard_library
standard_library.install_aliases()
from builtins import str
import tweepy, sys
```

```

import xml.dom.minidom, xml.sax.saxutils
from tweepy.utils import import_simplejson
json = import_simplejson()
from tweepy.models import Status
import logging
import splunk.entity as entity

import http.client
from socket import timeout
from tweepy.auth import BasicAuthHandler
from tweepy.api import API

#set up logging suitable for splunkd consumption
logging.root
logging.root.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(levelname)s %(message)s')
handler = logging.StreamHandler()
handler.setFormatter(formatter)
logging.root.addHandler(handler)

SCHEME = """<scheme>
  <title>Twitter</title>
  <description>Get data from Twitter.</description>
  <use_external_validation>true</use_external_validation>
  <streaming_mode>simple</streaming_mode>
  <endpoint>
    <args>
      <arg name="name">
        <title>Twitter feed name</title>
        <description>Name of the current feed using the user credentials supplied.</description>
      </arg>

      <arg name="username">
        <title>Twitter ID/Handle</title>
        <description>Your Twitter ID.</description>
      </arg>
      <arg name="password">
        <title>Password</title>
        <description>Your twitter password</description>
      </arg>
    </args>
  </endpoint>
</scheme>
"""

def do_scheme():
    print(SCHEME)

# prints XML error data to be consumed by Splunk
def print_error(s):
    print("<error><message>%s</message></error>" % xml.sax.saxutils.escape(s))

class SplunkListener( tweepy.StreamListener ):

    def on_data(self, data):
        super( SplunkListener, self ).on_data( data )
        twt = json.loads(data)
        if 'text' in twt:
            print(json.dumps(twt))
            return True

    def on_error(self, status_code):

```



```

        """Called when a non-200 status code is returned"""
        print('got error\n')
        print(status_code)
        logging.error("got error: %s" %(status_code))
        return False

    def on_timeout(self):
        """Called when stream connection times out"""
        print('got timeout')
        logging.info("Got a timeout")
        return

def validate_conf(config, key):
    if key not in config:
        raise Exception("Invalid configuration received from Splunk: key '%s' is missing." % key)

#read XML configuration passed from splunkd
def get_config():
    config = {}

    try:
        # read everything from stdin
        config_str = sys.stdin.read()

        # parse the config XML
        doc = xml.dom.minidom.parseString(config_str)
        root = doc.documentElement
        conf_node = root.getElementsByTagName("configuration")[0]
        if conf_node:
            logging.debug("XML: found configuration")
            stanza = conf_node.getElementsByTagName("stanza")[0]
            if stanza:
                stanza_name = stanza.getAttribute("name")
                if stanza_name:
                    logging.debug("XML: found stanza " + stanza_name)
                    config["name"] = stanza_name

                    params = stanza.getElementsByTagName("param")
                    for param in params:
                        param_name = param.getAttribute("name")
                        logging.debug("XML: found param '%s'" % param_name)
                        if param_name and param.firstChild and \
                            param.firstChild.nodeType == param.firstChild.TEXT_NODE:
                            data = param.firstChild.data
                            config[param_name] = data
                            logging.debug("XML: '%s' -> '%s'" % (param_name, data))

            checkpoint_node = root.getElementsByTagName("checkpoint_dir")[0]
            if checkpoint_node and checkpoint_node.firstChild and \
                checkpoint_node.firstChild.nodeType == checkpoint_node.firstChild.TEXT_NODE:
                config["checkpoint_dir"] = checkpoint_node.firstChild.data

        if not config:
            raise Exception("Invalid configuration received from Splunk.")

        # just some validation: make sure these keys are present (required)
        validate_conf(config, "name")
        validate_conf(config, "username")
        validate_conf(config, "password")
        validate_conf(config, "checkpoint_dir")
    except Exception as e:
        raise Exception("Error getting Splunk configuration via STDIN: %s" % str(e))

```

```

    return config

def get_validation_data():
    val_data = {}

    # read everything from stdin
    val_str = sys.stdin.read()

    # parse the validation XML
    doc = xml.dom.minidom.parseString(val_str)
    root = doc.documentElement

    logging.debug("XML: found items")
    item_node = root.getElementsByTagName("item")[0]
    if item_node:
        logging.debug("XML: found item")

        name = item_node.getAttribute("name")
        val_data["stanza"] = name

        params_node = item_node.getElementsByTagName("param")
        for param in params_node:
            name = param.getAttribute("name")
            logging.debug("Found param %s" % name)
            if name and param.firstChild and \
                param.firstChild.nodeType == param.firstChild.TEXT_NODE:
                val_data[name] = param.firstChild.data

    return val_data

# parse the twitter error string and extract the message
def get_twitter_error(s):
    try:
        doc = xml.dom.minidom.parseString(s)
        root = doc.documentElement
        messages = root.getElementsByTagName("Message")
        if messages and messages[0].firstChild and \
            messages[0].firstChild.nodeType == messages[0].firstChild.TEXT_NODE:
            return messages[0].firstChild.data
        return ""
    except xml.parsers.expat.ExpatError as e:
        return s

def validate_config(username,password):
    try:
        auth = BasicAuthHandler(username,password)
        headers = {}
        host = 'stream.twitter.com'
        url = '/1/statuses/sample.json?delimited=length'
        body = None
        timeout = 5.0
        auth.apply_auth(None,None, headers, None)
        conn = http.client.HTTPSConnection(host)
        conn.connect()
        conn.sock.settimeout(timeout)
        conn.request('POST', url, body, headers=headers)
        resp = conn.getresponse()
        if resp.status != 200:
            raise Exception("HTTP request to Twitter returned with status code %d (%s): %s" %
                (resp.status,resp.reason, get_twitter_error(resp.read())))
            logging.error("Invalid twitter credentials %s , %s" % (username,password))

```

```

        conn.close()
    except Exception as e:
        print_error("Invalid configuration specified: %s" % str(e))
        sys.exit(1)

def run():
    config = get_config()

    username=config["username"]
    password=config["password"]

    # Validate username and password before starting splunk listener.
    logging.debug("Credentials found: username = %s, password = %s" %(username,password))
    validate_config(username,password)

    listener = SplunkListener()
    stream = tweepy.Stream( username,password, listener )
    stream.sample()

if __name__ == '__main__':
    if len(sys.argv) > 1:
        if sys.argv[1] == "--scheme":
            do_scheme()
        elif sys.argv[1] == "--validate-arguments":
            if len(sys.argv)>3:
                validate_config(sys.argv[2],sys.argv[3])
            else:
                print('supply username and password')
        elif sys.argv[1] == "--test":
            print('No tests for the scheme present')
        else:
            print('You giveth weird arguments')
    else:
        # just request data from Twitter
        run()

    sys.exit(0)

```

### ***Twitter example spec file***

Place the following spec file in the following location:

```
$SPLUNK_HOME/etc/apps/twitter/README/inputs.conf.spec
```

### **inputs.conf.spec**

```

[twitter://default]
*This is how the Twitter app is configured

username = <value>
*This is the user's twitter username/handle

password = <value>
*This is the user's password used for logging into twitter

```

### ***Sample JSON input for the Twitter example***

Here is an example of the JSON input from Twitter that the Twitter example indexes:

```
{"contributors":null,"text":"@CraZiiBoSSx3 Yea ... Lo_Ok http://twitpic.com/19ksg2","created_at":"Fri Mar
```

```

19 18:41:17 +0000 2010", "truncated": false, "coordinates": null, "in_reply_to_screen
_name": "CraZiiBoSSx3", "favorited": false, "geo": null, "in_reply_to_status_id": 10735405186, "source": "<a
href=\"http://echofon.com/\"
rel=\"nofollow\">Echofon</a>", "place": null, "in_reply_to_user_id": 114199314, "user": {"created_at": "Mon Dec 21
23:01:05 +0000 2009", "profile_background_color": "0099B9", "favourites_count": 0, "lang": "en", "profile_text
_color": "3C3940", "location": "my location !", "following": null, "time_zone": "Central Time (US &
Canada)", "description": "Names GiiqqL3z; ;) ;
Unfollow", "statuses_count": 1685, "profile_link_color": "0099B9", "notifications": null, "profile_background_image
_url": "http://s.twimg.com/a/1268437273/images/themes/theme4/bg.gif", "contributors_enabled": false, "geo
_enabled": false, "profile_sidebar_fill_color": "95E8EC", "url": null, "profile_image_url": "http://a3.twimg.com/profile
_images/703836981/123_Kay_normal.jpg", "profile_background_tile": false, "protected": false, "profile_sidebar_border
_color": "5ED4DC", "screen_name": "123_Kay", "name": "~GLam Doll
GiiqqLez~", "verified": false, "followers_count": 77, "id": 98491606, "utc_offset": -21600, "friends
_count": 64}, {"id": 10735704604}

```

## S3 example

The S3 example provides for streaming data from the Amazon S3 data storage service. A more robust version of this capability is available in the Splunk Add-on for Amazon Web Services on Splunkbase.

Place the `s3.py` script in the following location in your Splunk installation:

```
$SPLUNK_HOME/etc/apps/s3/bin
```

**Note:** A script for modular inputs requires an `inputs.conf` spec file to operate correctly in Splunk Web. Refer to [Modular Inputs spec file](#) for information on creating the `inputs.conf` spec file.

`s3.py` reads files in various formats and streams data from the files for indexing by Splunk software. Specific areas of interest for modular inputs are the following:

- Connects to S3 services, providing an Access Key ID and a Secret Access Key
- Sets up logging to `splunkd.log`
- Provides an XML scheme for use by Splunk Manager
- Provides a `--scheme` argument that returns the XML scheme for the modular inputs
- Validates data returned from S3
- Specifies streaming mode as `xml`

### S3 example script

The following example script, `s3.py`, provides for streaming data from the Amazon S3 data storage service. `s3.py` is presented in its entirety below. Refer to [Scripts for modular inputs](#) for analysis of specific parts of the script. This script has been made cross-compatible with Python 2 and Python 3 using `python-future`.

#### s3.py

```

from __future__ import division
from __future__ import print_function
from future import standard_library
standard_library.install_aliases()
from builtins import str
from builtins import range
from past.utils import old_div
from builtins import object

```

```

import sys, time, os
import http.client, urllib.request, urllib.parse, urllib.error, hashlib, base64, hmac, urllib.parse, md5
import xml.dom.minidom, xml.sax.saxutils
import logging
import tarfile, gzip

ENDPOINT_HOST_PORT = "s3.amazonaws.com"

# set up logging suitable for splunkd consumption
logging.root
logging.root.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(levelname)s %(message)s')
handler = logging.StreamHandler()
handler.setFormatter(formatter)
logging.root.addHandler(handler)

SCHEME = """<scheme>
  <title>Amazon S3</title>
  <description>Get data from Amazon S3.</description>
  <use_external_validation>true</use_external_validation>
  <streaming_mode>xml</streaming_mode>

  <endpoint>
    <args>
      <arg name="name">
        <title>Resource name</title>
        <description>An S3 resource name without the leading s3://.
          For example, for s3://bucket/file.txt specify bucket/file.txt.
          You can also monitor a whole bucket (for example by specifying 'bucket'),
          or files within a sub-directory of a bucket
          (for example 'bucket/some/directory/'; note the trailing slash).
        </description>
      </arg>

      <arg name="key_id">
        <title>Key ID</title>
        <description>Your Amazon key ID.</description>
      </arg>

      <arg name="secret_key">
        <title>Secret key</title>
        <description>Your Amazon secret key.</description>
      </arg>
    </args>
  </endpoint>
</scheme>
"""

def string_to_sign(method, http_date, resource):
    # "$method\n$contentMD5\n$contentType\n$httpDate\n$xamzHeadersToSign$resource"
    return "%s\n\n\n%s\n%s" % (method, http_date, resource)

# returns "Authorization" header string
def get_auth_header_value(method, key_id, secret_key, http_date, resource):
    to_sign = string_to_sign(method, http_date, resource)
    logging.debug("String to sign=%s" % repr(to_sign))

    signature = base64.encodestring(hmac.new(str(secret_key), to_sign, hashlib.sha1).digest()).strip()

    return "AWS %s:%s" % (key_id, signature)

def put_header(conn, k, v):

```

```

        logging.debug("Adding header %s: %s" % (k, v))
        conn.putheader(k, v)

def gen_date_string():
    st = time.localtime()
    tm = time.mktime(st)
    return time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime(tm))

# query_string is expected to have been escaped by the caller
def get_http_connection(key_id, secret_key, bucket, obj, use_bucket_as_host = True, query_string = None):
    method = "GET"
    host = bucket + "." + ENDPOINT_HOST_PORT
    if not use_bucket_as_host:
        host = ENDPOINT_HOST_PORT

    conn = http.client.HTTPConnection(host)
    logging.info("Connecting to %s." % host)
    conn.connect()

    unescaped_path_to_sign = "/" + bucket + "/"
    unescaped_path_to_req = "/"
    if obj:
        unescaped_path_to_sign += obj
        unescaped_path_to_req += obj

    if not use_bucket_as_host:
        unescaped_path_to_req = unescaped_path_to_sign

    date_str = gen_date_string()

    path = urllib.parse.quote(unescaped_path_to_req)
    if query_string:
        path += query_string
    logging.debug("%s %s" % (method, path))
    conn.putrequest(method, path)
    put_header(conn, "Authorization", get_auth_header_value(method, key_id, \
        secret_key, date_str, unescaped_path_to_sign))
    put_header(conn, "Date", date_str)
    conn.endheaders()

    return conn

def log_response(resp):
    status, reason = resp.status, resp.reason
    s = "status=%s reason=\"%s\"" % (str(status), str(reason))
    if status == 200:
        logging.debug(s)
    else:
        logging.error(s)

# parse the amazon error string and extract the message
def get_amazon_error(s):
    try:
        doc = xml.dom.minidom.parseString(s)
        root = doc.documentElement
        messages = root.getElementsByTagName("Message")
        if messages and messages[0].firstChild and \
            messages[0].firstChild.nodeType == messages[0].firstChild.TEXT_NODE:
            return messages[0].firstChild.data
        return ""
    except xml.parsers.expat.ExpatError as e:
        return s

```

```

# prints XML error data to be consumed by Splunk
def print_error_old(s):
    impl = xml.dom.minidom.getDOMImplementation()
    doc = impl.createDocument(None, "message", None)
    top_element = doc.documentElement
    text = doc.createTextNode(s)
    top_element.appendChild(text)
    sys.stdout.write(doc.toxml())

# prints XML error data to be consumed by Splunk
def print_error(s):
    print("<error><message>%s</message></error>" % xml.sax.saxutils.escape(s))

def validate_conf(config, key):
    if key not in config:
        raise Exception("Invalid configuration received from Splunk: key '%s' is missing." % key)

# read XML configuration passed from splunkd
def get_config():
    config = {}

    try:
        # read everything from stdin
        config_str = sys.stdin.read()

        # parse the config XML
        doc = xml.dom.minidom.parseString(config_str)
        root = doc.documentElement
        conf_node = root.getElementsByTagName("configuration")[0]
        if conf_node:
            logging.debug("XML: found configuration")
            stanza = conf_node.getElementsByTagName("stanza")[0]
            if stanza:
                stanza_name = stanza.getAttribute("name")
                if stanza_name:
                    logging.debug("XML: found stanza " + stanza_name)
                    config["name"] = stanza_name

                params = stanza.getElementsByTagName("param")
                for param in params:
                    param_name = param.getAttribute("name")
                    logging.debug("XML: found param '%s'" % param_name)
                    if param_name and param.firstChild and \
                        param.firstChild.nodeType == param.firstChild.TEXT_NODE:
                        data = param.firstChild.data
                        config[param_name] = data
                        logging.debug("XML: '%s' -> '%s'" % (param_name, data))

            checkpoint_node = root.getElementsByTagName("checkpoint_dir")[0]
            if checkpoint_node and checkpoint_node.firstChild and \
                checkpoint_node.firstChild.nodeType == checkpoint_node.firstChild.TEXT_NODE:
                config["checkpoint_dir"] = checkpoint_node.firstChild.data

        if not config:
            raise Exception("Invalid configuration received from Splunk.")

        # just some validation: make sure these keys are present (required)
        validate_conf(config, "name")
        validate_conf(config, "key_id")
        validate_conf(config, "secret_key")
        validate_conf(config, "checkpoint_dir")

```

```

except Exception as e:
    raise Exception("Error getting Splunk configuration via STDIN: %s" % str(e))

return config

def read_from_s3_uri(url):
    u = urllib.parse.urlparse(str(url))
    bucket = u.netloc
    obj = None
    subdir = None
    if u.path:
        obj = u.path[1:] # trim the leading slash
        subdir = "/" + ".".join(obj.split("/")[:-1])
        if subdir:
            subdir += "/"
    logging.debug("Extracted from url=%s bucket=%s subdir=%s object=%s" % (url, bucket, subdir, obj))
    if not subdir:
        subdir = None
    if not obj:
        obj = None
    return (bucket, subdir, obj)

class HTTPResponseWrapper(object):
    def __init__(self, resp):
        self.resp = resp

def init_stream():
    sys.stdout.write("<stream>")

def fini_stream():
    sys.stdout.write("</stream>")

def send_data(source, buf):
    sys.stdout.write("<event unbroken=\"1\"><data>")
    sys.stdout.write(xml.sax.saxutils.escape(buf))
    sys.stdout.write("</data>\n<source>")
    sys.stdout.write(xml.sax.saxutils.escape(source))
    sys.stdout.write("</source></event>\n")

def send_done_key(source):
    sys.stdout.write("<event unbroken=\"1\"><source>")
    sys.stdout.write(xml.sax.saxutils.escape(source))
    sys.stdout.write("</source><done/></event>\n")

# returns a list of all objects from a bucket
def get_objs_from_bucket(key_id, secret_key, bucket, subdir = None):
    query_string = None
    if subdir:
        query_string = "?prefix=%s&delimiter=/" % urllib.parse.quote(subdir)
    conn = get_http_connection(key_id, secret_key, bucket, obj = None, query_string = query_string)
    resp = conn.getresponse()
    log_response(resp)
    if resp.status != 200:
        raise Exception("AWS HTTP request return status code %d (%s): %s" % \
            (resp.status, resp.reason, get_amazon_error(resp.read())))
    bucket_listing = resp.read()
    conn.close()

    # parse AWS's bucket listing response
    objs = []
    doc = xml.dom.minidom.parseString(bucket_listing)
    root = doc.documentElement

```



```

key_nodes = root.getElementsByTagName("Key")
for key in key_nodes:
    if key.firstChild.nodeType == key.firstChild.TEXT_NODE:
        objs.append(key.firstChild.data)

return objs

def get_encoded_file_path(config, url):
    # encode the URL (simply to make the file name recognizable)
    name = ""
    for i in range(len(url)):
        if url[i].isalnum():
            name += url[i]
        else:
            name += "_"

    # MD5 the URL
    m = md5.new()
    m.update(url)
    name += "_" + m.hexdigest()

    return os.path.join(config["checkpoint_dir"], name)

# returns true if the checkpoint file exists
def load_checkpoint(config, url):
    chk_file = get_encoded_file_path(config, url)
    # try to open this file
    try:
        open(chk_file, "r").close()
    except:
        # assume that this means the checkpoint it not there
        return False
    return True

# simply creates a checkpoint file indicating that the URL was checkpointed
def save_checkpoint(config, url):
    chk_file = get_encoded_file_path(config, url)
    # just create an empty file name
    logging.info("Checkpointing url=%s file=%s", url, chk_file)
    f = open(chk_file, "w")
    f.close()

def run():
    config = get_config()
    url = config["name"]
    bucket, subdir, obj = read_from_s3_uri(url)
    key_id = config["key_id"]
    secret_key = config["secret_key"]

    if obj and (not subdir or obj != subdir):
        # object-level URL provided (e.g. s3://bucket/object.txt) that does
        # not appear to be a directory (no ending slash)
        if not load_checkpoint(config, url):
            # there is no checkpoint for this URL: process
            init_stream()
            request_one_object(url, key_id, secret_key, bucket, obj)
            fini_stream()
            save_checkpoint(config, url)
        else:
            logging.info("URL %s already processed. Skipping.")
    else:

```

```

# bucket-level URL provided (e.g. s3://bucket), or a directory-level
# URL (e.g. s3://bucket/some/subdir/)
init_stream()
while True:
    logging.debug("Checking for objects in bucket %s" % bucket)
    objs = get_objs_from_bucket(key_id, secret_key, bucket, subdir)
    for o in objs:
        if subdir and not o.startswith(subdir):
            logging.debug("obj=%s does not start with %s. Skipping.", subdir)
            continue
        obj_url = "s3://" + bucket + "/" + o
        if not load_checkpoint(config, obj_url):
            logging.info("Processing %s" % obj_url)
            request_one_object(obj_url, key_id, secret_key, bucket, o)
            save_checkpoint(config, obj_url)

    # check every 60 seconds for new entries
    time.sleep(60)
fini_stream()

def request_one_object(url, key_id, secret_key, bucket, obj):
    assert bucket and obj

    conn = get_http_connection(key_id, secret_key, bucket, obj)
    resp = conn.getresponse()
    log_response(resp)
    if resp.status != 200:
        raise Exception("Amazon HTTP request to '%s' returned status code %d (%s): %s" % \
            (url, resp.status, resp.reason, get_amazon_error(resp.read())))

    translator = get_data_translator(url, resp)

    cur_src = ""
    buf = translator.read()
    bytes_read = len(buf)
    while len(buf) > 0:
        if cur_src and translator.source() != cur_src:
            send_done_key(cur_src)
            cur_src = translator.source()
        send_data(translator.source(), buf)
        buf = translator.read()
        bytes_read += len(buf)

    if cur_src:
        send_done_key(cur_src)

    translator.close()
    conn.close()
    sys.stdout.flush()

    logging.info("Done reading. Read bytes=%d", bytes_read)

# Handles file reading from tar archives. From the tarfile module:
# fileobj must support: read(), readline(), readlines(), seek() and tell().
class TarTranslator(object):
    def __init__(self, src, tar):
        self.tar = tar
        self.member = next(self.tar)
        self.member_f = self.tar.extractfile(self.member)
        self.translator = None
        self.base_source = src
        if self.member:

```

```

        self.src = self.base_source + ":" + self.member.name
        if self.member_f:
            self.translator = get_data_translator(self.src, self.member_f)

def read(self, sz = 8192):
    while True:
        while self.member and self.member_f is None:
            self.member = next(self.tar)
            if self.member:
                self.member_f = self.tar.extractfile(self.member)
                self.src = self.base_source + ":" + self.member.name
                self.translator = get_data_translator(self.src, self.member_f)

        if not self.member:
            return "" # done

        buf = self.translator.read(sz)
        if len(buf) > 0:
            return buf
        self.member_f = None
        self.translator = None

def close(self):
    self.tar.close()

def source(self):
    return self.src

class FileObjTranslator(object):
    def __init__(self, src, fileobj):
        self.src = src
        self.fileobj = fileobj

    def read(self, sz = 8192):
        return self.fileobj.read(sz)

    def close(self):
        return self.fileobj.close()

    def source(self):
        return self.src

class GzipFileTranslator(object):
    def __init__(self, src, fileobj):
        self.src = src
        self.fileobj = fileobj

    def read(self, sz = 8192):
        return self.fileobj.read(sz)

    def close(self):
        return self.fileobj.close()

    def source(self):
        return self.src

def get_data_translator(url, fileobj):
    if url.endswith(".tar"):
        return TarTranslator(url, tarfile.open(None, "r|", fileobj))
    elif url.endswith(".tar.gz") or url.endswith(".tgz"):
        return TarTranslator(url, tarfile.open(None, "r|gz", fileobj))
    elif url.endswith(".tar.bz2"):

```

```

        return TarTranslator(url, tarfile.open(None, "r|bz2", fileobj))
    elif url.endswith(".gz"):
        # it's lame that gzip.GzipFile requires tell() and seek(), and our
        # "fileobj" does not supply these; wrap this with the object that is
        # used by the tarfile module
        return GzipFileTranslator(url, tarfile._Stream("", "r", "gz", fileobj, tarfile.RECORDSIZE))
    else:
        return FileObjTranslator(url, fileobj)

def do_scheme():
    print(SCHEME)

def get_validation_data():
    val_data = {}

    # read everything from stdin
    val_str = sys.stdin.read()

    # parse the validation XML
    doc = xml.dom.minidom.parseString(val_str)
    root = doc.documentElement

    logging.debug("XML: found items")
    item_node = root.getElementsByTagName("item")[0]
    if item_node:
        logging.debug("XML: found item")

        name = item_node.getAttribute("name")
        val_data["stanza"] = name

        params_node = item_node.getElementsByTagName("param")
        for param in params_node:
            name = param.getAttribute("name")
            logging.debug("Found param %s" % name)
            if name and param.firstChild and \
                param.firstChild.nodeType == param.firstChild.TEXT_NODE:
                val_data[name] = param.firstChild.data

    return val_data

# make sure that the amazon credentials are good
def validate_arguments():
    val_data = get_validation_data()
    key_id = val_data["key_id"]
    secret_key = val_data["secret_key"]

    try:
        url = "s3://" + val_data["stanza"]
        bucket, subdir, obj = read_from_s3_uri(url)
        logging.debug("(%s', '%s', '%s') % (str(bucket), str(subdir), str(obj)))
        if subdir and subdir == obj:
            # monitoring a "sub-directory" within a bucket
            obj = None

            # see if there are any objects that would match that subdir
            all_objs = get_objs_from_bucket(key_id, secret_key, bucket, subdir)
            matches = False
            for o in all_objs:
                if o.startswith(subdir):
                    matches = True
                    break
            if not matches:

```

```

        raise Exception("No objects found inside s3://%s." % "/".join([bucket, subdir]))
    else:
        # use_bucket_as_host = False allows for better error checking:
        # AWS tends to return more helpfull error messages
        conn = get_http_connection(key_id, secret_key, bucket, obj, use_bucket_as_host = False)
        resp = conn.getresponse()
        log_response(resp)
        if old_div(resp.status, 100) == 3:
            # AWS may send a sometimes when it requires that the bucket
            # is part of the host: retry
            conn = get_http_connection(key_id, secret_key, bucket, obj, use_bucket_as_host = True)
            resp = conn.getresponse()
            log_response(resp)
        if resp.status != 200:
            raise Exception("Amazon returned HTTP status code %d (%s): %s" % (resp.status, resp.reason,
get_amazon_error(resp.read())))

    except Exception as e:
        print_error("Invalid configuration specified: %s" % str(e))
        sys.exit(1)

def usage():
    print("usage: %s [--scheme|--validate-arguments]")
    sys.exit(2)

def test():
    init_stream()
    send_data("src1", "test 1")
    send_data("src2", "test 2")
    send_done_key("src2")
    send_data("src3", "test 3")

if __name__ == '__main__':
    if len(sys.argv) > 1:
        if sys.argv[1] == "--scheme":
            do_scheme()
        elif sys.argv[1] == "--validate-arguments":
            validate_arguments()
        elif sys.argv[1] == "--test":
            test()
        else:
            usage()
    else:
        # just request data from S3
        run()

    sys.exit(0)

```

### ***S3 example spec file***

Place the following spec file in the following location:

```
$SPLUNK_HOME/etc/apps/s3/README/inputs.conf.spec
```

#### **inputs.conf.spec**

```

[s3://<name>]

key_id = <value>
* This is Amazon key ID.

```

```
secret_key = <value>  
* This is the secret key.
```

# Build scripted inputs

## Setting up a scripted input

This section describes how to set up a scripted input for an app. To illustrate the setup, it uses an example script that polls a database and writes the results to a file. A more detailed version of this example is in [Example script that polls a database](#). That topic provides details on the example, including code examples in Python and Java.

You can write any number and types of scripts in various scripting languages that perform various functions. This example shows the framework for a commonly found script. Adapt this framework according to your needs.

### Script to poll a database

This example script does the following.

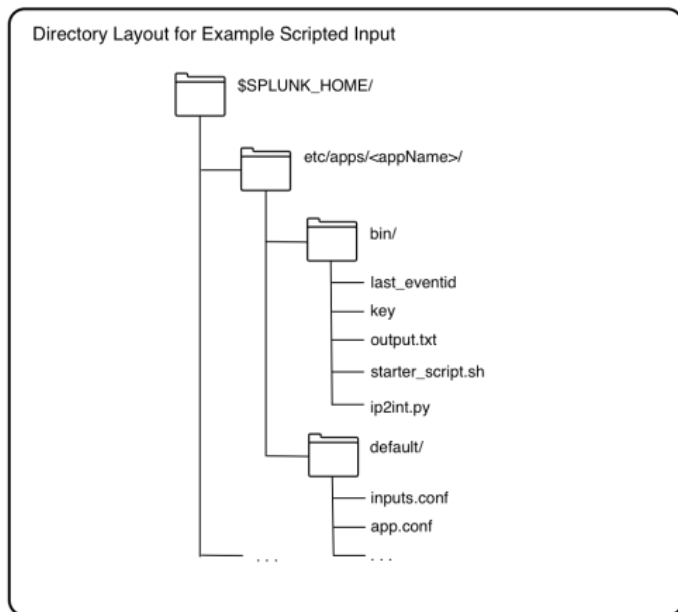
- Runs at a regular interval.
- Queries a database.
- Writes the output to a file in a format optimized for indexing.
- Splunk software indexes the file containing the results of the queries.

#### Directory structure

Place scripts in the `/bin` directory of your app.

```
$SPLUNK_HOME/etc/apps/<appName>/bin/
```

Here is the directory structure of the example script for this example. The directory structure for your app might differ.



## Script files

```
.../etc/apps/<appName>/bin/my_db_poll.py
```

This is the script that retrieves information from the database. This script does the following:

- Queries the database and writes the query result to file.
- Defines the format of output data.
- Accesses a database using credentials stored in key.
- Reads last\_eventid to determine the next event to read from the database.
- Queries the database at the next event and writes the output to a file.

```
.../etc/apps/<appName>/bin/starter_script.sh
```

Wrapper script that calls the `my_db_poll.py` script. In this example, it calls `my_db_poll.py` with the arguments needed to query the database.

In `.../etc/apps/<appName>/default/inputs.conf`, create a stanza that references this wrapper script. In this example, the stanza specifies how often to call the starter script to poll the database.

```
.../etc/apps/<appName>/bin/ip2int.py
```

A helper script to convert IP addresses from integer format to dotted format, and back. This is a type of helper script that formats data better for indexing. You often have helper scripts that aid the main script.

```
.../etc/apps/<appName>/bin/key
```

Text file containing username and password encoded in base64 using the python function `base64.b64encode()`. The Splunk Enterprise user has read and write access to this file.

Security for passwords is an issue when running scripts.

```
.../etc/apps/<appName>/bin/last_eventid
```

File containing a number for the last event received from the database. `my_db_poll.py` writes the last\_eventid after querying the database. The Splunk user has read and write access to this file.

```
'.../etc/apps/<appName>/bin/output.txt'
```

A single event from the script, for reference. `my_db_poll.py` writes the actual output from querying the database to another directory.

```
.../etc/apps/<appName>/default/inputs.conf
```

Configure scripted data input in `$SPLUNK_HOME/etc/<appName>/default/inputs.conf`. Use the local directory for the app to overwrite behavior defined in the default directory. Here is an example:

```
[script://$SPLUNK_HOME/etc/apps/<appName>/bin/starter_script.sh]
disabled = true # change to false to start the input, requires restart
host = # enter hostname here
index = main
interval = 30          #frequency to run the script, in seconds
source = my_db
```



```
sourcetype = my_db_data
```

```
$SPLUNK_HOME/etc/system/local/props.conf
```

Configure properties for the script in the Splunk Enterprise system `props.conf`.

```
[my_db_data]
TIME_PREFIX=[^\\|]+\\|
TIME_FORMAT=%Q
MAX_TIMESTAMP_LOOKAHEAD=10          #look ahead 10 characters
SHOULD_LINEMERGE=false
$SPLUNK_HOME/etc/system/local/transforms.conf
```

Define field transforms in `transforms.conf`.

```
[my_db_extractions]
DELIMS = "|"
FIELDS ="EventID", "AlertTime", "UserName", . . ."
```

## Writing reliable scripts

Here are some tips for creating reliable input scripts:

### Environment variables

Clear environment variables that can affect your script's operation. One environment variable that is likely to cause problems is the library path. The library path is most commonly known as `LD_LIBRARY_PATH` on Linux, Solaris, and FreeBSD. It is `DYLD_LIBRARY_PATH` on OS X, and `LIBPATH` on AIX.

If you are running external python software or using other python interpreters, consider clearing `PYTHONPATH`.

**Caution:** Changing `PYTHONPATH` may affect other installations of python.

On Windows platforms, the `SPLUNK_HOME` environment variable is set for you. Avoid changing this environment variable. Changing this variable may interfere with the functioning of Splunk Enterprise services.

### Python version

For best results, use the version of Python available from your Splunk Enterprise installation. Splunk Enterprise uses this version to execute system scripts. Use this version of Python to test your scripts.

Some Python libraries that your script requires may not be available in the Splunk platform's version of Python. In this case, you can copy the libraries to the same directory as the scripted input.

To run a script using the version of Python available from Splunk Enterprise:

```
$SPLUNK_HOME/bin/splunk cmd python <your_script>.py
```

### File paths in Python

Be careful when specifying platform-specific paths and relative paths.

## Platform-specific paths

When writing scripts in Python, avoid hard coding platform-specific file paths. Instead specify file paths that can be interpreted correctly on Windows, UNIX, and Mac platforms. For example, the following Python code launches `try.py`, which is in the `bin` directory of your app, and has been made cross-compatible with Python 2 and Python 3 using `python-future`.

```
from __future__ import print_function
import os
import subprocess

# Edit directory names here if appropriate
if os.name == 'nt':
    ## Full path to your Splunk installation
    splunk_home = 'C:\Program Files\Splunk'
    ## Full path to python executable
    python_bin = 'C:\Program Files (x86)\Python-2.7-32bit\python.exe'
else:
    ## Full path to your Splunk installation
    # For some reason:
    #splunk_home = '/appl/opt/splunk_fwd/'
    # For a sensible OS:
    splunk_home = '/opt/splunk'

    ## Full path to python executable
    # For Mac OS X:
    #python_bin = '/Library/Frameworks/Python.framework/Versions/2.7/bin/python'
    # For a sensible filesystem:
    python_bin = '/usr/bin/python'

try_script = os.path.join(splunk_home, 'etc', 'apps', 'your_app', 'bin', 'try.py')

print(subprocess.Popen([python_bin, try_script], stdout=subprocess.PIPE).communicate()[0])
```

## Relative paths

Avoid using relative paths in scripts. Python scripts do not use the current directory when resolving relative paths. For example, on \*nix platforms, relative paths are set relative to the root directory (/). The following example shows how to locate the `extract.conf` file, which is in the same directory as the script:

```
import os
import os.path

script_dirpath = os.path.dirname(os.path.join(os.getcwd(), __file__))

config_filepath = os.path.join(script_dirpath, 'extract.conf')
```

## Format script output

Format the output of a script so Splunk software can easily parse the data. Also, consider formatting data so it is more human-readable as well.

## Use the Common Information Model Add-on

The Common Information Model Add-on is based on the idea that you can break down most log files into three components: fields, event type tags, and host tags.

With these three components you can set up log files in a way that makes them easily processable and that normalizes non-compliant log files, forcing them to follow a similar schema. The Common Information Model Add-on organizes these fields and tags into categories and provides a separate **data model** for each category. You can use the CIM data models to test your data to ensure that it has been normalized correctly, and then report on it.

You can download the Common Information Model Add-on from Splunkbase [here](#). For a more in-depth overview of the CIM Add-on, see the *Common Information Model Add-on Manual*.

### ***Timestamp formats***

Time stamp the beginning of an event. There are several options for timestamp formats:

#### **RFC-822, RFC-3339**

These are standard timestamp formats for email headers and internet protocols. These formats provide an offset from GMT, and thus are unambiguous and more human-readable. RFC-822 and RFC-3339 formats can be handled with `%Z` in a `TIME_FORMAT` setting.

**RFC-822** Tue, 15 Feb 2011 14:11:01 -0800

**RFC-3339** 2011-02-15 14:11:01-08:00

#### **UTC**

UTC formatting may not be as human-readable as some of the other options. If the timestamp is epoch time, no configuration is necessary. Otherwise, requires a configuration in `props.conf` that declares the input as `TZ=UTC`.

#### **UTC**

2011-02-15T14:11:01-05:00

2011-02-15T14:11:01Z

#### **UTC converted to epoch time**

1297738860

### ***Multiline data and field names***

For multiline data, find a way to separate events.

- Write a distinctive initial line for a multiline event.
- Use a special *end of event* string to separate events. For example, use three newline characters to specify an end of an event. The event then includes any single or double newline characters.
- For multiline field values, place the field data inside quotes.
- Use an equals sign, `=`, or other separator to expose name/value pairs. For example, `key=value`.
- Configure your Splunk Enterprise instance to use other tokens that might exist in the data.
- Field names are case sensitive. For example the field names "message" and "Message" represent different fields. Be consistent when naming fields.

## Create a setup page to configure scripted inputs

If you are packaging an app or add-on for distribution, consider creating a setup page that allows users to interactively provide configuration settings for access to local scripted input resources. For more information, see [Enable first-run configuration with setup pages in Splunk Cloud Platform or Splunk Enterprise on the Splunk Developer Portal](#).

## Save state across invocations of the script

Scripts often need to checkpoint their work so subsequent invocations can pick up from where they left off. For example, save the last ID read from a database, mark the line and column read from a text file, or otherwise note the last input read. (See [Example script that polls a database](#).)

You can check point either the index or the script. When check pointing data, keep in mind that the following things are not tied together as a transaction:

- Writing out checkpoint files
- Fully writing data into the pipe between the script and splunkd
- splunkd completely writing out the data into the index

Thus, in the case of hard crashes, it's hard to know if the data the script has acquired has been properly indexed. Here are some of the choices you have:

**Search Splunk index** One strategy is to have the scripted input search in the Splunk index to find the last relevant event. This is reasonable in an infrequently-launched script, such as one that is launched every 5 or 10 minutes, or at launch time for a script which launches once and stays running indefinitely.

**Maintain independent check point** Because there is some delay between data being fed to the Splunk platform and the data becoming searchable, a frequently run scripted input must maintain its own checkpoint independent of the index.

**Choose a scenario** If the script always believes its own checkpoint, data may not be indexed on splunkd or system crash. If the index search is believed, some data may be indexed multiple times on splunkd or system crash. You need to choose which scenario you best fits your needs.

## Accessing secured services

Use proper security measures for scripts that need credentials to access secured resources. Here are a few suggestions on how to provide secure access. However, no method is foolproof, so think carefully about your use case and design secure access appropriately:

- Restrict which users can access the app or add-on on disk.
- Create and use credentials specific to the script, with the minimum permissions required to access the data.
- Avoid putting literal passwords in scripts or passing the password as a command line argument, making it visible to all local processes with operating system access.
- Use Splunk Enterprise to encrypt passwords. You can create an app set up page that allows users to enter passwords. See the setup page example with user credentials on the Splunk developer portal. The user can enter a password in plain text, which is stored in the credential stanza in apps.conf. Alternatively, you can specify a python script to securely provide access.

**Caution:** Splunk Enterprise assembles a secret using locally available random seeds to encrypt passwords stored in configuration files. This method provides modest security against disclosure of passwords from admins with local disk read capability. However, it is not an adequate protection for privileged accounts.

## Concurrency issues for scripted inputs

Be careful scheduling two copies of a script running at any given time. Splunk Enterprise detects if another instance of the script is running, and does not launch a new instance if this is the case. For example, if you have a script scheduled to execute every 60 seconds, and a particular invocation takes 140 seconds, Splunk Enterprise detects this and does not launch a new instance until after the long-running instance completes.

At times you may want to run multiple copies of a script, for example to poll independent databases. For these cases, design your scripts so they can handle multiple servers. Also, design your script so that multiple copies can exist (for example, use two app directories for the script).

Alternatively, you could have separate scripts using the same source type.

## Troubleshooting scheduled scripts

Splunk Enterprise logs exceptions thrown by scheduled scripts to the `splunkd.log` file, located here:

```
$SPLUNK_HOME/var/log/splunk/splunkd.log
```

Check `splunkd.log` first if expected events do not appear in the expected index after scheduling the scripted input.

## Shutdown and restart issues

Keep these shutdown and restart issues in mind when designing your scripts:

### ***Output at least one event at a time***

This makes it easier to avoid reading a partial event if the script is terminated or crashes. Splunk Enterprise expects events to complete in a timely manner, and has built-in time-outs to prevent truncated or incomplete events.

Configure the pipe `fd` as line-buffered, or `write()` full events at once. Be sure the events are flushed: `line buffered/unbuffered/fflush()`

### ***Output relatively small batches of events***

Fetching thousands of event over a few minutes and then outputting them all at once increases the risk of losing data due to a restart. Additionally, outputting small batches of events means your data is searchable sooner and improves script transparency.

## Example script that polls a database

Here is an example of a scripted input that polls a database. In the configuration for the script, you specify the interval at which the script runs.

**Note:** No script can be a "one size fits all." The purpose of this example is to provide a basic framework that you modify and customize for your specific purposes. This script polls a database and writes the records retrieved to stdout. The data queries, connection, authentication, and processing of the query have been simplified.

This example script does the following:

- Builds a query to extract 1000 records from a database
- Connects to a database
- Stores the key to the database as an eventID.
- Writes the last eventID retrieved from the database to file to track which events have been indexed.
- Executes the query and writes the results to stdout for the Splunk platform to index.

## Pseudo-code for the example script

```
# Script to poll a database
#
# Reads 1000 records from a database,
# writes them to stdout for indexing by splunk,
# tracks last event read
#
# SQL Query information:
#
# Microsoft SQL Server syntax
# SELECT TOP 1000 eventID, transactionID, transactionStatus FROM table
#   WHERE eventID > lastEventID ORDER BY eventID
#
#
# MySQL syntax
# SELECT eventID, transactionID, transactionStatus FROM table
#   WHERE eventID > lastEventID LIMIT 1000 ORDER BY eventID
#
#
# Oracle syntax
# SELECT eventID, transactionID, transactionStatus FROM table
#   WHERE eventID > lastEventID AND ROWNUM <= 1000 ORDER BY eventID
#
# =====
# Database Fields
# =====
#
# eventID                autoincrement unsigned
# transactionId char      8
# transactionStatus varchar 32
#
# =====
# Sample Data
# =====
#
# 1 A1756202    submitted
# 2 C1756213    acknowledged
# 3 A1756202    rejected
# 4 N1756754    submitted
# 5 C1756213    completed
```

```
import needed files
```

```
define SQL query
```

```
define SQL connection information
```

```
db server address
db user
db pw
db name
```

```
define path to file that holds eventID of last record read
```

```

last_eventid_filepath

read eventID from last_eventid file

connect to database

execute SQL query

write query results to stdout

close db connection

update eventID in last_eventid file

```

## Script example, poll a database (Python)

Here is a python version of the database poll example. The code has been simplified for readability and does not necessarily represent best coding practices. Please modify according to your needs.

The Python version of the example accesses a Microsoft SQL Server database. It assumes you have all the necessary libraries referenced in the script.

This example requires the following:

- pymssql language extension
- FreeTDS 0.63 or newer (\*nix and Mac OS X platforms only)

This script has been made cross-compatible with Python 2 and Python 3 using python-future.

### hello\_db\_poll\_script.py

```

#!/usr/bin/python

from __future__ import print_function
from builtins import str
import _mssql
import os
import sys
from time import localtime, strftime
import time

sql_server = "SQLserver" #Address to database server
database = "hello_db_database"
sql_uname = "splunk_user"
sql_pw = "changeme"
columns = 'TOP 1000 eventID, transactionID, transactionStatus'
table = 'hello_table'

countkey = 'eventID'

last_eventid_filepath = "" # user supplies correct path

# Open file containing the last event ID and get the last record read
last_eventid = 0;
if os.path.isfile(last_eventid_filepath):
    try:
        last_eventid_file = open(last_eventid_filepath, 'r')
        last_eventid = int(last_eventid_file.readline())
        last_eventid_file.close()

```

```

# Catch the exception. Real exception handler would be more robust
except IOError:
    sys.stderr.write('Error: failed to read last_eventid file, ' + last_eventid_filepath + '\n')
    sys.exit(2)
else:
    sys.stderr.write('Error: ' + last_eventid_filepath + ' file not found! Starting from zero. \n')

# Fetch 1000 rows starting from the last event read
# SELECT TOP 1000 eventID, transactionID, transactionStatus FROM table WHERE eventID > lastEventID ORDER BY
eventID
sql_query = 'SELECT ' + columns + ' FROM ' + table + ' WHERE ' + countkey + ' > ' + str(last_eventid) + '
ORDER BY ' + countkey

try:
    conn = _mssql.connect(sql_server, sql_uname, sql_pw, database)
    conn.execute_query(sql_query)
    # timestamp the returned data
    indexTime = "[" + strftime("%m/%d/%Y %H:%M:%S %p %Z", localtime()) + "]"
    for row in conn:
        print("%s eventID=%s, transactionID=%s, transactionStatus=%s" % (indexTime, row['eventID'],
row['transactionID'], row['transactionStatus']))

        this_last_eventid = row['eventID']

# Catch the exception. Real exception handler would be more robust
except _mssql.MssqlDatabaseException as e:
    sys.stderr.write('Database Connection Error!\n')
    sys.exit(2)

finally:
    conn.close()

if this_last_eventid > 0:
    try:
        last_eventid_file = open(last_eventid_filepath, 'w')
        last_eventid_file.write(this_last_eventid)
        last_eventid_file.close()
    # Catch the exception. Real exception handler would be more robust
    except IOError:
        sys.stderr.write('Error writing last_eventid to file: ' + last_eventid_filepath + '\n')
        sys.exit(2)

```



# Customize Splunk Web

## Customization options and caching

### Options for customization

There are several styling, behavior, and text customization options available for different components of the Splunk platform.

To learn about	See
Customizing the login page	<a href="#">Customize the login page</a>
Using CSS or JavaScript files to <ul style="list-style-type: none"><li>• Customize individual dashboards</li><li>• Customize all dashboards in an app</li></ul>	<a href="#">Customize dashboard styling and behavior</a>
Configuring UI internationalization	<a href="#">UI internationalization</a>
Adding HTML or images to a dashboard	The <html> Simple XML element reference in <i>Dashboards and Visualizations</i>
Customizing integrated PDFs	Additional configuration options for integrated PDF generation in the <i>Reporting Manual</i>

### Clear client and server assets caches after customization

Static assets for apps are cached on the client and server side. If you update an item in the `/appserver/static` directory, you can see changes by clearing both the client and server caches. Clear the browser cache to update the client side.

To clear the server cache, use one of the following options:

- Clear Splunk Enterprise's client-side Splunk Web resources using: `_bump`:  
`http://<host:mport>/<locale_string>/_bump`
- Clear all splunkd registered EAI handlers that support reload using: `debug/refresh`:  
`http://<host:mport>/debug/refresh`
- Restart `splunkd`. For more information, see *Start and stop Splunk Enterprise* in the *Admin Manual*.
- Set `cacheEntriesLimit=0` in `web.conf`. This setting is recommended only for development use cases and not for production. For more information, see the `web.conf` spec file in the *Admin Manual*.

## Customize the login page

Splunk Enterprise users can customize Splunk Web login page components.

## Add custom text

If you are using Splunk Enterprise, you can customize the Splunk platform login page with plain or HTML formatted text.

### Prerequisite

Review the `login_content` setting details in the `web.conf` spec file.

### Steps

1. Check the `$SPLUNK_HOME/etc/system/local/` directory for a `web.conf` file. Use one of the following options.

File already exists in the directory	File does not exist in the directory
Locate the <code>[settings]</code> stanza in the file.	<ol style="list-style-type: none"><li>1. Create a new file called <code>web.conf</code> in the directory.</li><li>2. Add a <code>[settings]</code> stanza in the file.</li></ol>

2. In the local `web.conf` file, add or edit the `login_content` string under the `[settings]` stanza.  
`login_content = <content_string>`

Ensure that the text and formatting are no longer than one line in the configuration file. See the custom text example.

3. Restart the Splunk instance to view the change.

### Example

```
[settings]
login_content = This is a <b>production server</b>.<br>For expensive searches try: <a
href="http://server2:8080">server2</a>
```

## Customize the login page background

If you are using Splunk Enterprise, you can use Splunk Web to customize the login page background. Display a custom image, a default image, or no image.

You can also configure the login page background image using the `loginCustomBackgroundImage` and `loginBackgroundImageOption` settings in `$SPLUNK_HOME/etc/system/local/web.conf`. See the `web.conf` spec file for more information.

**Note:** Image alignment and scaling customizations are not available. Depending on the browser window size, the background image appearance can vary.

### Prerequisites

To edit the login page background, your role must hold the `edit_server` capability.

If you are adding a custom image, make sure that the image file meets the following requirements:

- Use a `.jpg`, `.jpeg`, or `.png` formatted file.
- A landscape oriented image is recommended.
- The maximum file size is 20MB.
- The suggested minimum image size is 1024x640 pixels.

## Steps

1. Log into the Splunk instance and navigate to **Settings > System > Server Settings > Login Background**.
2. Select one of the following options.

Background option	Description
Custom image	To use a custom background, upload the image file and click <b>Choose</b> .
Default image	Use the default background image.
No image	Do not display an image on the login page.

3. Use the **Preview** screen to preview the login page customization.
4. Click **Save**.
5. Restart the Splunk instance to view the changes.

## Add a custom logo

If you are using Splunk Enterprise, you can customize the login page logo.

### Prerequisites

- The maximum image size is 485px wide and 100px high. If the image exceeds these limits, the image is automatically resized.
- Review the `loginCustomLogo` setting details in the `web.conf` spec file.

## Steps

1. (Optional) If you are using an image file, put it into the following directory location.  
`$(SPLUNK_HOME)/etc/apps/<app_name>/appserver/static/logo`
2. Check the `$(SPLUNK_HOME)/etc/system/local/` directory for a `web.conf` file. Use one of the following options.

File already exists in the directory	File does not exist in the directory
Locate the <code>[settings]</code> stanza in the file.	<ol style="list-style-type: none"><li>1. Create a new file called <code>web.conf</code> in the directory.</li><li>2. Add a <code>[settings]</code> stanza in the file.</li></ol>

3. In the local `web.conf` file, add or edit the `loginCustomLogo` setting under the `[settings]` stanza. Indicate the `loginCustomLogo` file path or an image URL.
4. Restart the Splunk instance to view the change.

## Use a custom favicon

Splunk Enterprise users can add a custom favicon to use across Splunk Web.

### Prerequisites

- Review the `customFavicon` setting details in the `web.conf` spec file.
- Make sure that the favicon image file meets the following requirements.
  - ◆ Use only an `.ico` formatted file.
  - ◆ The image must be square. No other image shapes are supported.

## Steps

1. Put the image file into the following directory location.  
`$(SPLUNK_HOME)/etc/apps/<app_name>/appserver/static/customfavicon`
2. Check the `$(SPLUNK_HOME)/etc/system/local/` directory for a `web.conf` file. Use one of the following options.

File already exists in the directory	File does not exist in the directory
Locate the <code>[settings]</code> stanza in the file.	<ol style="list-style-type: none"><li>1. Create a new file called <code>web.conf</code> in the directory.</li><li>2. Add a <code>[settings]</code> stanza in the file.</li></ol>

3. In the local `web.conf` file, add or edit the `customFavicon` setting under the `[settings]` stanza. Indicate the `customFavicon` file path.
4. Restart the Splunk instance to view the change.

## Customize dashboard styling and behavior

Customize dashboard appearance and behavior using custom `.css` and `.js` files. You can customize a specific dashboard within an app or all dashboards in a particular app.

Including custom JavaScript files can cause dashboard rendering issues. You might see a warning about custom scripts when you open a dashboard in Edit mode.

### Customize styling and behavior for one dashboard

#### Create custom files

To customize a specific dashboard, start by creating one or more `.css` or `.js` files to define styling and behavior.

Depending on the app to which the dashboard belongs, place these files in the app's `appserver/static` directory, located here.

```
$(SPLUNK_HOME)/etc/apps/<app_name>/appserver/static
```

For example, to customize styling and behavior for a dashboard in **Search and Reporting**, use this directory path.

```
$(SPLUNK_HOME)/etc/apps/search/appserver/static
```

#### Add custom files to the dashboard

When custom files are in the app's `appserver/static` directory, add them to the dashboard. Use the following syntax.

```
<dashboard stylesheet="<style_filename>.css" script="<script_filename>.js">
```

You can use several custom files for a dashboard. For multiple `.css` or `.js` files, use the following syntax.

```
<dashboard stylesheet="<style_filename1>.css, <style_filename2>.css" script="<script_filename1>.js, <script_filename2>.js">
```

**Note:** Forms have the `<form>` root element in Simple XML instead of `<dashboard>`. Use `<form stylesheet="...">` if you are adding a custom file to a form.

#### Add custom files from a different app to a dashboard

You can add custom files from one app's `appserver/static` directory to another app's dashboard. Use this syntax to indicate the other app context for custom files.

```
<dashboard stylesheet="<app_name>:<style_filename>.css" script="<app_name>:<script_filename>.js">
```

For example, to refer to files located in the **Search and Reporting** app context, use this syntax.

```
<dashboard stylesheet="search:my_custom_styles.css" script="search:my_custom_script.js">
```

**Note:** Ensure that custom files exist in the indicated app's `appserver/static` directory. Dependency checking and warning messages are not supported when files are not found.

## Customize styling and behavior for all dashboards in an app

Dashboards automatically load `dashboard.js` and `dashboard.css` from the `appserver/static` directory. To customize styling and behavior for all dashboards in an app, create one or both of the following files.

- `dashboard.js`
- `dashboard.css`

Place the files in the following directory.

```
$SPLUNK_HOME/etc/apps/<app_name>/appserver/static
```

---

## UI internationalization

Internationalize the Splunk Web user interface.

- Translate text generated by Python code, JavaScript code, views, menus and Mako templates.
- Set language/locale specific alternatives for static resources such as images, CSS, other media.
- Create new languages or locales.
- Format times, dates and other numerical strings.

## Splunk software translation

Splunk software uses the language settings for the browser where you are accessing Splunk Web. You can change the browser language settings to see the user interface in another language.

Locale strings indicate the language and location that Splunk software uses to translate the user interface. Typically, a locale string consists of two lowercase letters and two uppercase letters linked by an underscore. For example, `en_US` means American English while `en_GB` means British English.

Splunk software first tries to find an exact match for the full locale string but falls back to the language specifier if settings are not available for the full locale. For example, translations for `fr` answer to requests for `fr_CA` and `fr_FR` (French,

Canada and France respectively).

In addition to language, translation also addresses the formatting of dates, times, numbers, and other localized settings.

## Configuration

Splunk software uses the `gettext` internationalization and localization (i18n) system.

### Steps

1. Create a directory for the locale. For example, to create the fictional locale `mz`, create the following directory.  
`$SPLUNK_HOME/lib/python2.7/site-packages/splunk/appserver/mrsparkle/locale/mz_MZ/LC_MESSAGES/`
2. Load the following `messages.pot` file into your PO editor.  
`$SPLUNK_HOME/lib/python2.7/site-packages/splunk/appserver/mrsparkle/locale/messages.pot`
3. Use the PO editor to translate any strings that you want to localize. Save the file as `messages.po` in the directory you created in the previous step. The PO editor also saves a `messages.mo` file, which is the machine readable version of the PO file.
4. Restart the Splunk instance. No other configuration file edits are required. Splunk software detects the new language files when it restarts.

### Localization files

The Splunk platform stores localization information at the following location.

```
$SPLUNK_HOME/lib/python<version>/site-packages/splunk/appserver/mrsparkle/locale
```

This directory contains the following items.

- `messages.pot`: Holds the strings to translate. You can use a PO editor to edit these files.
- `<locale_string>`: Directory containing localization files for the locale specified by `<locale_string>` (for example, `ja_JP`).
- `<locale_string>/LC_MESSAGES/messages.po`: Contains the source strings specified for localization in `messages.pot`. Using a PO editor, provide the translations for these strings.
- `<locale_string>/LC_MESSAGES/messages.mo`: Machine readable version of `messages.po`. Splunk software uses this file to find translated strings. The PO editor creates the file for you when it creates the `messages.po` file.

### Localize dates and numbers

You can format numbers and dates to the standards of a locale without translating any text. Create a directory for the locale whose numbers and dates you want to format. Copy the contents of the `en_US` directory to the target locale directory.

#### Example

Enable localization of numbers and dates for the `de_CH` locale (German – Switzerland).

Create the following target directory for the `de_CH` locale.

```
$SPLUNK_HOME/lib/python2.7/site-packages/splunk/appserver/mrsparkle/locale/de_CH
```

Copy the contents of the following directory.

```
$SPLUNK_HOME/lib/python2.7/site-packages/splunk/appserver/mrsparkle/locale/en_US
```

Copy the contents from the `en_US` directory into the `de_CH` directory.

## Translate Apps

You can use `gettext` to translate apps. Most apps must be translated in their own locale subdirectory.

Apps that ship with the Splunk platform are automatically extracted and their text is included in the core `messages.pot` file. You do not need to handle them separately.

To extract the strings from an installed app and make the strings ready for translation in a PO editor, run the following extraction command on the command line.

```
> splunk extract i18n -app <app_name>
```

This creates a `locale/` subdirectory in the app root directory and populates it with a `messages.pot` file.

Follow the [steps above](#) to translate the strings within the app. When using views from a different app, the new `messages.pot` file contains the strings for these views.

## Locale-specific resources

The Splunk platform stores static resources such as images, CSS files, and other media as subdirectories at the following location.

```
$SPLUNK_HOME/share/splunk/search_mrsparkle/exposed/
```

When serving these resources, Splunk software checks to see whether a localized version of the resource is available before falling back to the default resource. For example, if your locale is set to `fr_FR`, Splunk software searches for the logo image file in the following order.

- `exposed/img/skins/default/logo-mrsparkle-fr_FR.gif`
- `exposed/img/skins/default/logo-mrsparkle-fr.gif`
- `exposed/img/skins/default/logo-mrsparkle.gif`

Splunk software follows the same path to load HTML templates (including any views) that define each page in the UI. This can be useful for languages that require a modified layout that CSS alone cannot accommodate (right to left text for example).

## **Building custom apps**

### **Developer resources**

To learn about designing and building custom apps, see the [Splunk Developer Portal](#).