Splunk Developer's Guide

Design, implement, and publish custom Splunk applications and add-ons following best practices

Kyle Smith
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 4 'Basic Views and Dashboards'
- A synopsis of the book’s content
- More information on Splunk Developer's Guide

About the Author

Kyle Smith is a self-proclaimed geek and has been working with Splunk extensively since 2010. He enjoys integrating Splunk with new sources of data and types of visualizations. He has spoken numerous times at the Splunk User Conference (most recently in 2014 on lesser-known search commands) and is an active contributor to the Splunk Answers community as well as on the #splunk IRC channel. He has published several Splunk Apps and add-ons to Splunk base, the Splunk community's premier Apps and add-ons publishing platform. He has worked in both higher education and the private industry, most recently as an infrastructure analyst at a Fortune 400 company. He lives in Central Pennsylvania with his family.
Splunk Developer's Guide

Splunk is awesome! You can not only consume virtually any data, but also extend and integrate Splunk with virtually any external system. Splunk uses sets of configurations referred to as applications or add-ons, which are the primary focus of this book. Leveraging these applications and add-ons is what gives Splunk its unique ability to extend, learn, analyze, and visualize information.

Splunk helps users determine the root cause of a failure, get a quick overview of system health, and take a deep dive into SQL statements and messages, just to name a few. Aggregation and centralization of log and event management are a growing trend in the big data sphere. By leveraging the combined intelligence gathered from correlating disparate sets of data, businesses or individuals can make data-based decisions. This book will help Splunk developers, or even simply curious end users, develop different methods of consuming new data and design new types of visualizations. Also it simply offers tips and tricks that help the software development life cycle.

What This Book Covers

Chapter 1, Application Design Fundamentals, covers fundamental questions and considerations before diving into an App or add-on configuration.

Chapter 2, Creating Applications, discusses the basic methods of App and add-on creation, along with an explanation of the structure of an App or add-on.

Chapter 3, Enhancing Applications, shows you a few different configurations that help enrich your data with Splunk knowledge objects, along with some basic App and add-on branding guidelines.

Chapter 4, Basic Views and Dashboards, goes over the basics of SimpleXML dashboard creation and development.

Chapter 5, The Splunk Web Framework, details the various SplunkJS stack components, and shows examples of how to use them within an HTML dashboard.

Chapter 6, Advanced Integrations and Development, reviews modular inputs, data models, the KV store, and modular D3 visualizations.

Chapter 7, Packaging Applications, lists the items needed to package an App or add-on, getting it ready for publishing.

Chapter 8, Publishing Applications, describes step by step how to upload an App to Splunk base, and includes some information on Splunk's great support community.
Basic Views and Dashboards

In this chapter, we will use a web interface to start creating basic views and dashboards that will help visualize the data we have been collecting and enhancing. Knowing what data you have and how it is arranged is paramount for quickly building views and dashboards. It also gives you a place to start, and shows which parts of your data get displayed on which dashboard. Yeah, you can go ahead and just start creating dashboards, views, and saved searches, but with a little bit of planning, you can avoid refactoring multiple times. As with any project, setting the scope of what your App is going to address is a key factor in its making. After we tackle the broad questions on what to include and how, we will cover the following topics: using a saved search in a panel, and the different types of visualizations available by default in Splunk. We will use some of these saved searches and visualizations to create a SimpleXML dashboard, and then convert that dashboard into a SimpleXML form. We will also briefly discuss HTML dashboards, but those will be covered in depth in the next chapter. Hence, we will only show you how to convert existing dashboards and create new ones from scratch.

Knowing your data

Knowing your data is paramount when creating dashboards and views. Being familiar with your data gives you a deeper understanding of the context and a greater insight into the different nuances of data visualization. For example, you will never use time series data in a pie chart, right? Knowing that you have time series data before trying to create a chart will speed up the process of designing and creating the dashboards you will need in your App. Another very valid thought process entails you to think about which data fits which data visualization it's more than just the typical pie versus time-based line charts. This can include such things as choosing between bar and column charts, when to use stacked bars, or whether you should use 100 percent stacked bars. Some of these will be answered only once you work with the visualization previews available in Splunk.
Let's start with a quick exercise. Let's assume that you are collecting Kbps metrics for your internet connection. You collect these metrics every 5 minutes and index them in Splunk. What kind of graph would best leverage time-based statistics? A line graph? Area graphs? Yes! Bar graphs? Pie graphs? No! Bar and pie graphs do not adequately show time-based data. But, on the other hand, if you were trying to determine the percentage of users using the Internet who were in specific ranges of age, should you use a line graph? Probably not, as this type of data will fit better within a pie graph or bar graph. This is because you are interested in statistics over all of your time range, not just in specific time intervals (such as 5 minutes).

**Modules available**

Let's get a quick overview of the types of charts available in Splunk 6.2. These are charts that can be used directly from the dashboard editor. The built-in visualizations are as follows: Line, Area, Column, Bar, Pie, Scatter, Bubble, Single Value, Radial Gauge, Filler Gauge, Marker Gauge, and Map. Each of these displays data differently, and, obviously, not all data will fit into all types of visualizations. Splunk helps you know your data and offers recommendations based on the search you have completed, as shown in the following screenshot. This search used a `timechart` command, so the recommended charts were those that deal well with time-based data—Line, Area, and Column.
As you can see, these are the native charts in Splunk. Additional modules are available or can be created, but these are the basic visualizations.

**SimpleXML dashboard**

Okay, so now we get to play around with some visualizations. Let's create a dashboard with some visualizations of our https://meh.com/ data. For now, we will limit ourselves to the native visualizations. We will start with the `meh_products` lookup that we have generated. Each product is listed as either New or Refurbished, so let's see how that makes a chart:

```
| inputlookup meh_products | top product_condition |
```

This will be our basic search that pulls the `meh_products` lookup, which is kept populated by the saved search we had written earlier. We are only interested in knowing the distribution of product conditions over the entire time range, so we will chose a pie chart from the dropdown. This gives us a pie chart like this one:

As you can see, at the time of writing this book, and with the amount of data collected, the top condition at 90.909% is New. Now, let's save this as a panel in a new dashboard. We will call the dashboard Overview. We do this by clicking on the **Save As** dropdown in the top-right corner of the screen. You would want to save it as a **Dashboard Panel**. This pops up a dialog to create a new dashboard, or append this visualization to an existing dashboard. We want to create a new dashboard. Fill in the form, as shown in the following screenshot, and then click on **Save**.
Since we are developing a full-blown App, we want to make sure that this dashboard is created in the app, and not in our user folder. So, make sure that **Shared in App** is selected.

This will create the dashboard and place that pie chart on it. Once you have this created, you need a way to quickly get to the dashboard. Let's edit the `nav.xml` file and add the name of this dashboard. Add the following line to the XML file, right under the `nav` tag of the file. Hit the `debug/refresh` endpoint to load the change:

```xml
<view name="overview" default="true" />
```

When you navigate to the app, this **Overview** dashboard will be the first to load. Let's add some more data visualizations, shall we? You can add as many as you want, but keep in mind that if more visualizations are added, more searches will need to be dispatched, and this may decrease search performance of the search head.
This is what we came up with; isn't it pretty?

All of these data points are calculated from the `meh_products` lookup file, and they cover all the items located therein. But, what if we want time-based data? Let's create another dashboard. This one will use the summaries of the `https://meh.com/polls` that we have been collecting. We won't go step by step as we create the time poll data dashboard; the steps are fairly similar, as before. Don't forget to add your new dashboard to the navigation! The following screenshot gives us the poll results:
Wow! That was quick. Do you see how we are using line charts and bar charts to display time-based data? But wait! The data looks funny. That is because this dashboard is showing everything over all the time for which we collected data. A new poll starts everyday, with new answers, so this dashboard currently shows the aggregate of the entire collection. Not really useful in the grand scheme of things, which leads us to... SimpleXML forms!

**SimpleXML forms**

A SimpleXML form allows you to convert a SimpleXML dashboard into a dashboard with inputs. This allows you to narrow down the set of information from all time to a defined range of time. Let's go ahead and convert this dashboard into a form. Find the **Edit** dropdown in the top-right corner of the dashboard, and then choose **Edit Panels**. Here, you will see **Add Input**. Click on that and select **Time**. There are also other options available, as you can see in this screenshot:

![SimpleXML form options](image)

Clicking on **Time** will add a dropdown in the top-left corner of the dashboard. Now, it is considered a form. While still in edit mode, select the time range picker and set the value to **Today**. This will load only today's data into the form that was created. To leverage this new restriction, you must update each panel. Edit each panel search, click on the **Edit Search String** item, and update **Time Range Scope** to **Shared Time Picker (field1)**. This will allow each panel to use the time range picked in the dropdown at the top of the form. Once you have edited all the panels, click on the **Done** button in the top-right corner, and then refresh the page. You will notice that the data is cleaner and displays only a single poll's information. This allows the information to be displayed cleanly and with context. Remember that chart from the bottom of the dashboard? Ugly, wasn't it? Very cluttered and not useful; now that same chart is nicely clean and displays data in two dimensions.
The data here uses a bar chart, with each answer as a different color (and column). This chart gives you a time-based representation of poll results. Time is shown from top (midnight) to bottom. Each different color represents a different poll response, and from left to right is the calculated difference in the number of votes (essentially, it is the distance from one data point to the next in the summary), which shows the number of additional votes that that answer received from the previous time frame in the summary.

Let's look a little closer at a few bars. The following screenshot shows a close-up of the far left answer. As you can see, there is a lot of activity in the first few hours of the day, and it steadily decreases over the course of the day.

While this might not be a common use case for bar charts, it represents the data differently, while giving context to each data point.
Now that we have the form responding to a specific time range, let's clean up the search activity. As it currently stands, each panel in this form executes a new search. That's not very effective, especially when the dataset could be massive. So, let's talk about post-processing. What is post-processing? Essentially, we will define a single search element for the dashboard. Then, we will use those search results in every other panel of the dashboard—one search transformed multiple times. We start by editing the source of the form, located under the Edit dropdown (find the Edit Source option). Now, add in this XML code, right before the first <row> tag:

```xml
<search id="baseSearch">
  <query>
    index=summary category=meh_poll | table _time PollTitle votes answer
  </query>
  <earliest>${field1.earliest}</earliest>
  <latest>${field1.latest}</latest>
</search>
```

This sets up the base search for all other panels that reference the baseSearch ID. Did you notice the ${field1.earliest} token? It refers to the time picker dropdown that was added earlier in the section where we converted this dashboard into a form. Don't worry about it for now; just add it in there, as shown. Now, find your first panel. You will see a <search> tag underneath the <title> tag. Add base="baseSearch" to the <search> tag, and change the <query> tag to have only the stats command. You should end up with something like this:

```xml
<search base="baseSearch">
  <query>stats values(PollTitle)</query>
</search>
```

Do this for each panel in the form. Once you're done, click on the Save button. Now each panel within your form will load as soon as the main search is done. In this way, you can speed up the rendering of the form without extolling a huge search cost on your infrastructure. Another great optimization trick is to use a scheduled search, summary index, or data model as the base search.

**Warning**

Dashboards with an excessive number of searches can cause the search performance to degrade, causing issues within a production environment. Use optimization tricks and configurations to keep the searches running clean.
Let's circle back to that $field1.earliest$ item we saw earlier. This is a tokenized field. Splunk tokenizes the variable, allowing dynamic drill-downs and conditional displays and capturing inputs. You can then reference the "tokenized" element using the $variable$ name. We are going to implement one of these to dynamically limit the results shown, based on the number of votes for an answer. First, click on the Add Input button (after entering edit mode, of course), and add a Text box. Once it is placed in the field set in the top-left corner, click on the little pencil icon to edit the field. Enter the values shown in this screenshot:

<table>
<thead>
<tr>
<th>T Text</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio</td>
<td>Label</td>
</tr>
</tbody>
</table>
| Dropdown
| Checkbox | Search on Change | ✓ |
| Multiselect |
| Time | | |

Click on the **Apply** button and then on the **Done** button to finish creating the panel.

Now, you would want to add that limitation to your searches, so pick a search that you want to restrict. Add the token name (surrounded by $) to the search as if you were specifying a field. For example, we might only want the answers with more than 100 votes to be displayed on our time-based line chart. The original post process query is this:

```
where isint(votes) | timechart span=1h max(votes) as Votes by answer useother=f
```
We will update it to the following:

```
where isint(votes) AND votes > $min_votes$ | timechart span=1h
max(votes) as Votes by answer useother=f | eval threshold = $min_votes$.
```

Save the panel and refresh the form. The default value is 0, which is meant to include all results, but once it is changed to 100, the chart displays only the answers that have more than 100 votes; in effect, showing when each answer crossed the threshold.

There are many, many other options available for SimpleXML forms and dashboards. Since the use cases far outweigh the space required to detail them, we'll let you discover and explore the specifications for SimpleXML yourself, and instead turn towards HTML dashboards.

**HTML dashboards**

HTML dashboards are simply that. These are dashboards that have been either written from scratch in HTML and JavaScript, or converted from a SimpleXML dashboard. The primary purpose of an HTML dashboard is to give the developer the greatest degree of freedom possible. Anything that you can do in HTML with CSS and JavaScript can most likely be done in an HTML dashboard as well, with the added benefit of being able to use the entirety of Splunk information.

Let's go ahead and convert a SimpleXML dashboard into an HTML dashboard. Let's use the Overview dashboard we created earlier. To convert an existing dashboard, simply navigate to it in the web UI, then click on the **Edit** dropdown, and then click on **Convert to HTML**. A dialog will show up, and you will want to click **Replace Current** in this instance. If you're creating HTML dashboards for the first time, you may want to use **Create New** until you feel comfortable with the process.
Warning
Replacing a dashboard is permanent; there is no undo feature. You will also lose the ability to edit the panels directly in the editor, and you will not be able to generate a PDF file from the dashboard.

If you are ready, click on Convert Dashboard. You can then click on View to view the dashboard. What has changed? Nothing visually, really. The real magic happens behind the scenes, and we will delve into this in the next chapter. But, say you wanted to create a new HTML dashboard from scratch. The easiest way is to create a blank SimpleXML dashboard and then convert it inline. This will prepopulate all of the HTML and JavaScript you will need to get started with building a dashboard from scratch.
Summary

In this chapter, we covered some basic aspects of dashboards and view creation. Making sure you know your data is paramount for successful execution in data visualization. Splunk helps you discover your data, but you need to understand your data to display it properly. Not all data types and results fit into all chart types.

We also covered creating a SimpleXML dashboard, and then converting that dashboard into a form. SimpleXML dashboards give you ease of creation and allow you to manipulate them within the web interface. This is an excellent option for quick displays of data, without diving too deep into the code. We also briefly introduced HTML dashboards. There are several advantages of HTML dashboards, especially the added benefits of using the entire HTML, CSS and JavaScript stacks.

In our next chapter, we will start the joys of HTML dashboarding, and see how to use the SplunkJS stack effectively.
Where to buy this book


Click here for ordering and shipping details.