Dashboard Dirty Tricks (of the simpleXML type)

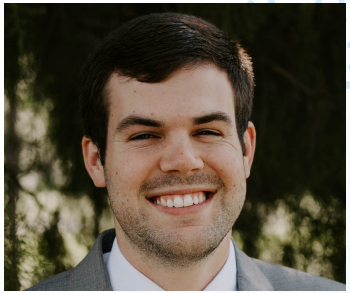
Father & Son Greetings!

Gregg Woodcock

President of Splunxter

We specialize in taking any Splunk project or infrastructure from "good enough" to "**Out Of This World**"!

Woodcock@Splunxter.com



Noah Woodcock

Sr Splunk Engineer at Splunxter

Full time with Splunk for last 6 years (started as a teenager!) We are ALL Splunk, ALL the time!

Noah@Splunxter.com

BSIDES splunk>

“

But isn’t SimpleXML going away? – (everybody in the audience right now)

Splunk dashboard frameworks come & go...

But SimpleXML will outlive them all!

BSIDES splunk>

EVERYBODY needs a Token Debugger

Conventional Approach: use "**Developer Gadgets App for Splunk**":
https://splunkbase.splunk.com/app/3689/

Sneaky Solution: use "**Splunk Dashboard Examples**":
https://splunkbase.splunk.com/app/1603/

1. Copy your dashboard XML into paste buffer.
2. Switch context into the "Dashboard Examples" app and create a new dashboard there (so that the javascript is active).
3. Paste your dashboard XML into the new dashboard.
4. Change the first XML line to this:
   <form script="event_token_browser.js">
5. Profit.

The magic of <SHIFT>-<CTRL>-E

Situation: You are using macros extensively (as you should) and it is getting very confusing to debug

Sneaky Solution: (Almost) Always sandwich your macro meat with a "TRACE" comment bun like this:

```
 rename TRACE AS "BEGIN OF MACRO 'conditional_outputlookup (1) '"

...Your normal SPL stuff here ...

| rename TRACE AS "END OF MACRO 'conditional_outputlookup(1)'"
```

Customizing Maps is NOT EASY!

Situation: Splunk Maps (cluster and choropleth) are great, but confusing. Normally you would find everything that you need at **docs.splunk.com** but…

Sneaky Solution: Use these alternative resources instead:

- Michael Porath's "**Use Custom Polygons in Choropleth Maps**" blog: https://www.splunk.com/en_us/blog/tips-and-tricks/use-custom-polygons-in-your-choropleth-maps.html
- Sideview's Nick Mealy's 3-part blog series: https://sideviewapps.com/2871/custom-maps/
- Mark Sivill's "**Mother Trucker**" presentation from .conf2018: https://www.splunk.com/en_us/resources/videos/visualize-this-mother-trucker.html
- **Pro Tip #1: Don't use KMZ** format unless you must; try to stick to single-file KML (KMZ = zipped KML + other files) so that "|inputlookup YourLookupKML" works (will not work for KMZ)
- **Pro Tip #2:** Don't bother with built-in mapping, start out with "**Maps+**" (you will eventually end up there anyway and it is WAY easier and VERY different): https://splunkbase.splunk.com/app/3124/

Silly Map zoom/tile default settings quirk

Situation: Why can't I zoom my map down to the level of detail that I'd like?

Sneaky Solution: You must configure a set of map tiles:

- Click "**Edit**"
- Click the painbtrush icon to open the "**Format**" dialog
- Click on the "**Tiles**" tab/section
- Update the "**Max Zoom**" value to something bigger than the default of "**7**".
- Examine the "**URL setting**", which starts out blank. Directly below this setting is a note with a sample URL to use for tiles.
- Copy this URL and paste it into the "**URL setting**" text box.
- Click "**Save**".

Multiple token sets on change of search

Situation: You need to set 2 tokens, each with 2 possible values based on search results.
Conventional Approach: Logic Multiplexing Matrix (scales poorly):

```
<change>
  <condition match="$result.count1$!=0 AND $result.count2$!=0">
    <set token="showtab1">t1</set>
    <set token="showtab2">t2</set>
  </condition>
  <condition match="$result.count1$!=0>
    <set token="showtab1">t1</set>
    <unset token="showtab2"></unset>
  <condition match="$result.count2$!=0">
    <set token="showtab2">t2</set>
    <unset token="showtab1"></unset>
  </condition>
  <condition>
    <unset token="showtab1"></unset>
    <unset token="showtab2"></unset>
  </condition>
</change>
```

Multiple token sets on change of search

Situation: You need to set 2 tokens, each with 2 possible values based on search results.
Undocumented Sneaky Solution: Multiple Change Sections (scales well):

```
<change>
  <condition match="$result.count1$!=0">
    <set token="showtab1">t1</set>
  </condition>
  <condition>
    <unset token="showtab1"></unset>
  </condition>
</change>
<change>
  <condition match="$result.count2$!=0">
    <set token="showtab2">t2</set>
  </condition>
  <condition>
    <unset token="showtab2"></unset>
  </condition>
</change>
```

Splunk stubbornly (re)sorts lexicographically

Situation: You need fields sorted your way (e.g. severity first, then lexicographically)

Undocumented Sneaky Solution: Leading Whitespace Padding

```
...
| rename COMMENT1of2 AS "Sort putting worst assets first/leftmost"
| rename COMMENT2of2 AS "This assumes highest severity == worst"
| eval asset = printf("%*s", len(asset) + severity, asset)
| chart count BY status asset
...
```

Reference: https://answers.splunk.com/answers/529004/is-there-a-way-to-display-more-than-20-charts-at-a.html

Splunk natively supports emojis!

Situation: You need quick and easy graphics

Undocumented Sneaky Solution: Use emojis!

Because your OS and browser support ANSI emojis, so does Splunk, including glyphs like "check mark" and "cross mark" that have broad application. Explore with this app:
Emoji: https://splunkbase.splunk.com/app/4273/

Or just cut and paste what you need from this site:
http://unicode.org/emoji/charts/full-emoji-list.html#2600

```
| makeresults
| eval status="PASS FAIL UP DOWN OOPS" | makemv status | mvexpand status
| eval icon = case(
    status=="PASS", "✅",
    status=="FAIL", "❌",
    status=="UP", "⬆️",
    status=="DOWN", "⬇️",
    true(), "❓")
```

Gibberish colors like "0xaBcdEF“, "#123456"

Situation: Your boss keeps changing color shades on you and updates take forever

Conventional Approach: ID, search, replace, repeat:

<option name="**mapping.fieldColors**">{
"BUG": **#000000**,
"CRITICAL": **#D35400**,
"WARN": **#F1C40F**,
"INFO": **#2ECC71**
"NULL": #C4C4C0}

Gibberish colors like "0xaBcdEF“, "#123456"

Undocumented Sneaky Approach:You’re a coder, use init & static variables:

```
<init>
   <set token="COLOR_GREEN">#2ECC71</set>
   <set token="COLOR_BLUE">#3498DB</set>
   <set token="COLOR_YELLOW">#F1C40F</set>
   <set token="COLOR_RED">#D35400</set>
   <set token="COLOR_GRAY">#C4C4C0</set>
   <set token="COLOR_BLACK">#000000</set>
   <set token="COLOR_DEBUG_ERROR">$COLOR_BLACK$</set>
</init>
...
<option name="mapping.fieldColors">{
"BUG":       $COLOR_DEBUG_ERROR$,
"CRITICAL": $COLOR_RED$,
"WARN":     $COLOR_YELLOW$,
"INFO":      $COLOR_GREEN$,
"NULL":      $COLOR_GRAY$}
```

Field colors, as documented, appear static

Situation: You need field colors assigned dynamically

```
<search id="siteFTWbase" base="siteGYRbase">
  <query>where site="FTW" | head 1 | table color</query>
    <done>
      <condition match="$result.color$==&amp;quot;GREEN&amp;quot;">
        <set token="FTW_COLOR">$COLOR_GREEN$</set></condition>
      <condition match="$result.color$==&amp;quot;BLUE&amp;quot;">
        <set token="FTW_COLOR">$COLOR_BLUE$</set></condition>
      <condition match="$result.color$==&amp;quot;BLACK&amp;quot;">
        <set token="FTW_COLOR">$COLOR_BLACK$</set></condition>
      <condition match="$result.color$==&amp;quot;YELLOW&amp;quot;">
        <set token="FTW_COLOR">$COLOR_YELLOW$</set></condition>
      <condition match="$result.color$==&amp;quot;RED&amp;quot;">
        <set token="FTW_COLOR">$COLOR_RED$</set></condition>
      <condition>
        <set token="FTW_COLOR">$COLOR_DEBUG_ERROR$</set>
      </condition>
    </done>
  </query>
</search>
```

Undocumented Sneaky Approach: Use dynamic tokens

```
<option name="mapping.fieldColors">{
FTW_ASSET_1:$FTW_COLOR$,
FTW_ASSET_2:$FTW_COLOR$,
MAR_ASSET_1:$MAR_COLOR$,
MAR_ASSET_2:$MAR_COLOR$}</option>
```

Sharing across multiple dashboards: macros!

Situation: Your need to change the colors across ALL dashboards at once

Undocumented Sneaky Approach: Use hidden base search to load color tokens from macros!

MACRO NAME	MACRO DEFINITION	MACRO NAME	MACRO DEFINITION
XML_COLOR_**BLACK**,	"#**000000**"	XML_COLOR_**BLUE**,	"#**3498DB**"
XML_COLOR_DEBUG_ERROR	`XML_COLOR_**BLACK**`	XML_COLOR_**GREEN**,	"#**2ECC71**"
XML_COLOR_**RED**,	"#**D35400**"	XML_COLOR_**YELLOW**,	"#**F1C40F**"

```
<search id="MacroToColors">
  <query>|rest/servicesNS/-/$env:app$/configs/conf-macros/
| regex title="^XML_COLOR_"
| map max_searches=99 search="|makeresults | eval $$title$$ = `$$title$$`"
| eval color = "code" | stats first(*) AS * BY color | table color *</query>
  <earliest>-1s</earliest>
  <latest>now</latest>
  <done>
    <set token="COLOR_GREEN">$result.XML_COLOR_GREEN$</set>
    <set token="COLOR_BLUE">$result.XML_COLOR_BLUE$</set>
    <set token="COLOR_YELLOW">$result.XML_COLOR_YELLOW$</set>
    <set token="COLOR_RED">$result.XML_COLOR_RED$</set>
    <set token="COLOR_BLACK">$result.XML_COLOR_BLACK$</set>
    <set token="COLOR_DEBUG_ERROR">$result.XML_COLOR_DEBUG_ERROR$</set>
  </done>
</search>
```

Leveraging populating searches for ad-hoc use

Situation: You have a Summary Index or Lookup Creator search but need "nowish" data, too

Undocumented Sneaky Solution:Use "| savedsearch" with tricky selection logic:

```
 <search id="timepickerHistoryBase" depends="$site$">
  <query>|makeresults | rename COMMENT AS "YOU MUST LIST ALL OF THE TOKENS USED IN 'HistorySearch' IN THE "depends" section
here or they will not be updated when changed!“ | addinfo | eval info_max_time = coalesce(info_max_+ 0, now())
| foreach info_min_time info_max_time [eval b&lt;&lt;FIELD&gt;&gt; = &lt;&lt;FIELD&gt;&gt; | bin span=1d b&lt;&lt;FIELD&gt;&gt;]
| eval WHICH = if(binfo_min_time != binfo_max_time OR info_min_time &lt; relative_time(now(), "-365d"), "CSV", "SPL")</query>
  <earliest>$timepicker.earliest$</earliest>
  <latest>$timepicker.latest$</latest>
  <done>
   <condition match="$result.WHICH$==&quot;CSV&quot;">
    <set token="HistorySearch">|inputlookup Last365DaysDirtyTricks.csv| addinfo | rex field=asset "(?&lt;asset_number&gt;\d+)$"
| where asset_number &lt;= "$LastNumberToken$"| eval info_max_time = coalesce(info_max_+ 0, now())
| where _time &gt;= info_min_time AND _time &lt;= info_max_time| fields - info_*</set>
   </condition>
   <condition>
    <set token="HistorySearch">|savedsearch "Daily Populating: Last 365 Days Dirty Tricks Lookup"| rex field=asset
"(?&lt;asset_number&gt;\d+)$"
| where asset_number &lt;= "$LastNumberToken$"</set>
   </condition>
  </done>
 </search>
 ...
 <search><query>$HistorySearch$ | ...]</query>
      <earliest>$timepicker.earliest$</earliest>
      <latest>$timepicker.latest$</latest>
 </search>
```

Multiselect with/without “All”

Situation: You have many possible multiselect values AND you use “ALL” but sometimes it gets added in the middle and users can’t find it and get confused. Why can’t it do something smarter?

Undocumented Sneaky Approach: Do it manually!

```
<input type="multiselect" token="site" searchWhenChanged="true">
    <label>Site(s)</label>
    <choice value="*">ALL</choice>
    <choice value="FTW">FTW</choice>
    <choice value="MAR">MAR</choice>
   <change>
    <eval token="form.site">case(
     mvcount($form.site$) == 2 AND mvindex($form.site$, 0) == "*",
       mvindex($form.site$, 1),
     mvfind($form.site$, "^\\*$") == mvcount($form.site$) - 1,
       "*",
     true(),
       $form.site$,
     false(),
         "YOU MUST SET 'searchWhenChanged' to 'true'!")</eval>
   </change>
    <default>*</default>
    <initialValue>*</initialValue
```

That's all folks!

BSIDES splunk>