20 minutes to add a dynamic table of contents in a Surveyor report

Introduction

Surveyor is a fantastic tool, no doubt about that. It excels at extracting and presenting data out of virtually any database. However, when it comes to organizing the reports produced, it lacks some features. As a matter of example, it' relatively easy to design a report (to stick to the Surveyor terminology, I'll call them *views* as from now, where the word *report* designates the effective output) that shows the CPU utilization on each system in a group, but depending on the number of systems in that group, the report size might easily reach hundreds of pages.

Whether this is a good practice or not is another debate. What I want to illustrate here is how to add a non-intrusive table of contents in the report.

By *table of contents* (TOC), I mean a set of hyperlinks which I can click to jump immediately at the right place instead of scrolling through dozens of pages to find the info I'm looking for.

By *non-intrusive*, I want this TOC immediately available under the elbow when I need it, and out of sight when I don't.

Lastly, of course must this TOC be dynamically generated (versus hard-coded). I agree it'll be a bit more code, but the code will keep operational for much longer.

This applies to HTML output only: unfortunately, Surveyor's PDF and Word export engines are not powerful enough to mimic this behavior. So far...

This is what it'll look like: fig. 1 shows the report with the TOC open.

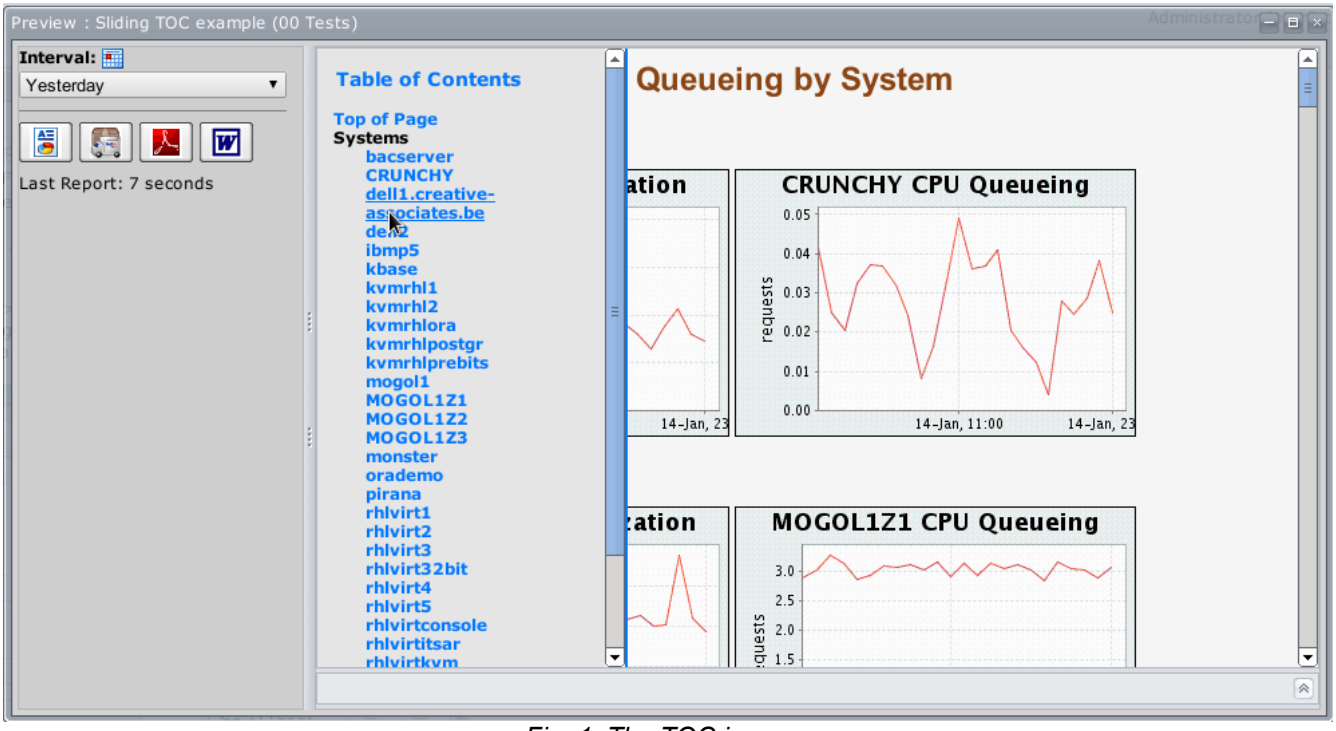

Fig. 1: The TOC is open

By mousing out of the TOC, it slides left; the only sign of the TOC presence is this blue line at the most left of the report. I can continue to scroll through the report as usually but whenever I need the TOC, I just have to move the mouse to the left of the report: automatically, the TOC shows back again. If I click on any hyperlink, the browser scrolls to the required position.

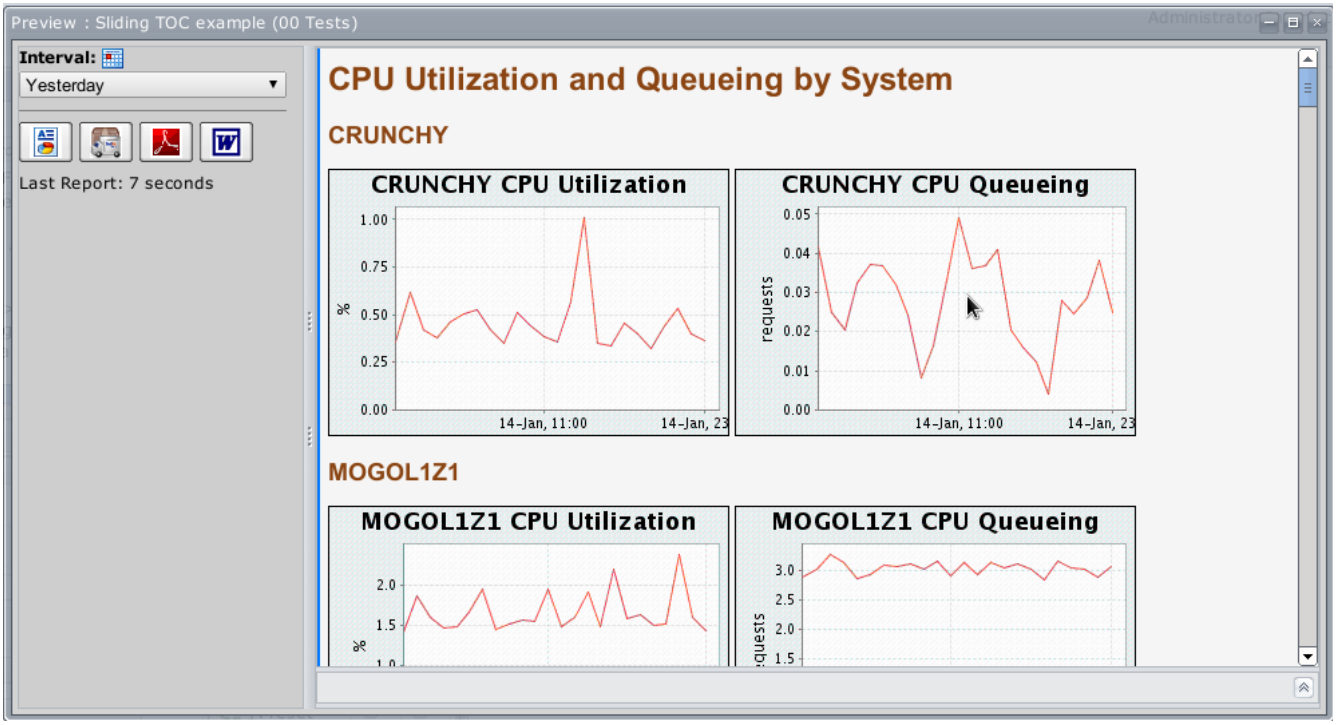

Fig. 2: The TOC is hidden, since the mouse moved outside the TOC area

The final version of this view is available for download: unzip in a temporary folder and import the pkg file inside. The zip file also contains a demo report you can play with: right-click on the demoToc.html file and open in a web browser. To minimize the dependencies in the demo report, the zip file contains the jQuery library, which is required.

The material described in this document is totally free of use, including the download-able file. Use and abuse at will.

However, I encourage you to build it yourself step by step as described here. This will help to understand the way it works and enhance it as new ideas raise.

In the first part of this howto, we'll design a basiv view.

Next, we'll add the basic dynamic TOC.

Lastly, we'll improve it.

Anyway, go as you please and have fun.

Design the report

First, we need a view to apply the TOC on.

The report itself is rather simple: I want a report showing the CPU utilization and CPU queue for each system in Surveyor. The charts are placed side by side under a title showing the system name, as seen in *fig.2*.

Step 1: Create the view

You need now to create a new view. In Surveyor, open the *Edit views* panel and right-click on a category. In the pop-up menu, select *New view…* Give your new view a name and press the *OK* button.

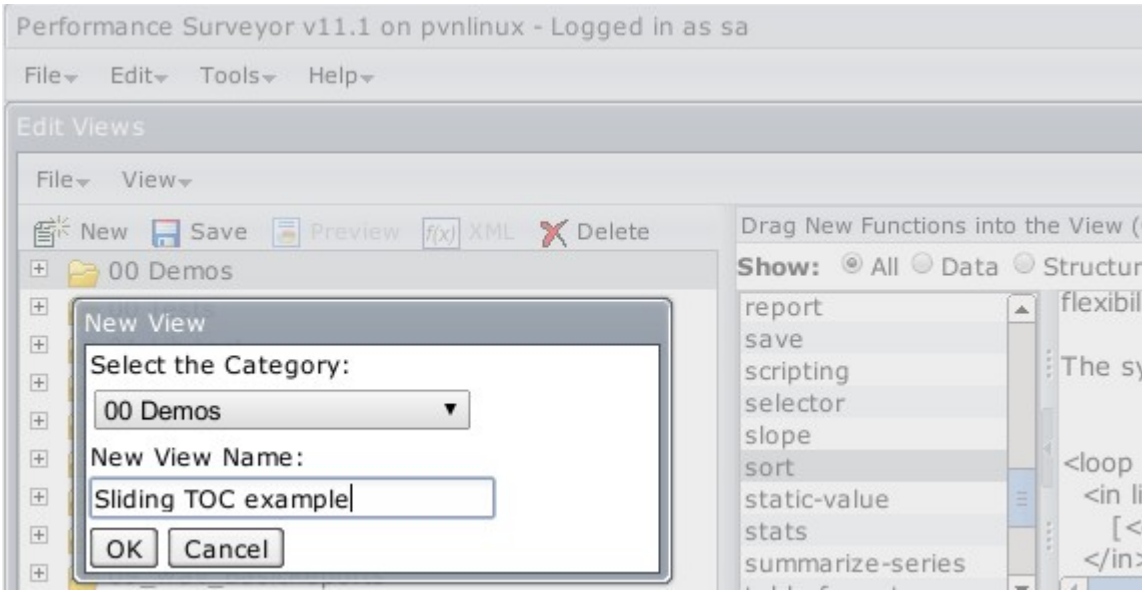

Fig. 3: Create a new view

Step 2: design the view contents

Open up the category. You have now to drop a few functions in the view.

1. Drag-drop a report in the view
2. Drag-drop a variable in the report added at (1)
3. Drag-drop a loop-in-do in the report added at (1)
4. Expand the loop function
5. Drag-drop a report in the do element
6. Drag-drop a chart in the report added at (5)

7. Drag-drop a get-data in the chart added at (6)
8. Drag-drop a second chart in the report added at (5)
9. Drag-drop a get-data in the chart added at (8)
10. Drag-drop a report in the report added at (1)

Here's how your view should look like now:

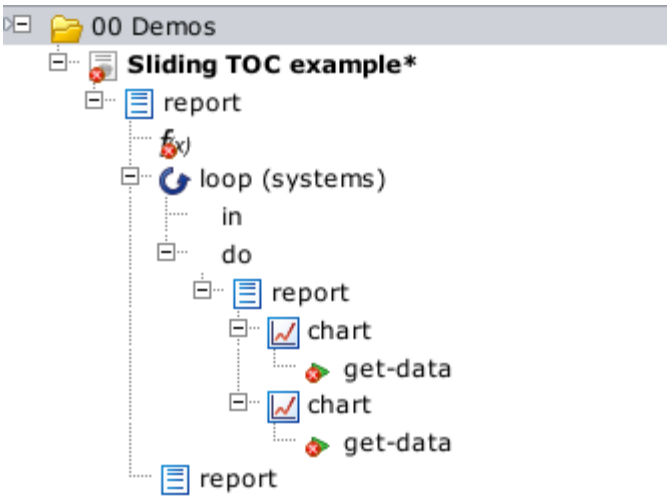

Fig. 4: The view structure

Step 3: re-label the functions

We'll first re-label these functions to be more accurate in the further explanations. To re-label a view, click it in the tree then select the *common* tab. Press the button in regard to the *label* item. An input box appears: that's the place where you must give the new label.

Now, rename according to *fig.5*. Please pay attention to use the same labels, as I'll now refer to the functions by their label.

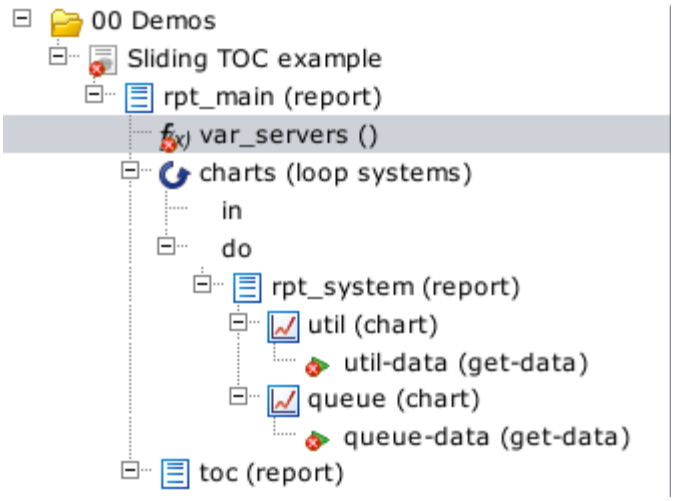

Fig. 5: Re-labeled functions

It is now time to save the view.

Step 4: fill in the function parameters

We have now a view structure, filled in with functions but these functions are not configured yet, so let's do that by now. I'll start with the deepest ones and climb the structure levels one by one. However, two of them will be left empty for now: *var_servers* and *toc*: they are dedicated to the TOC, which I'll cover in the next topic. For now, I'll concentrate on designing the report itself.

Function label	**Property**	**Value**	**Input mode**
util-data	**interval+**	☐ $iterator Yesterday	Prompt
	metric+	CPU Utilization	Preset
	summary-type+	Off	Preset
	system+	☑ $iterator	Preset
queue-data	**interval+**	☐ $iterator Yesterday	Prompt
	metric+	CPU Run Queue Length	Preset
	summary-type+	Off	Preset
	system+	☑ $iterator	Preset
util	legend	None	Preset
	shapes	false	Preset
	size	300x200	Preset
	title	${system} CPU Utilization	Preset

queue	legend	None	Preset
	shapes	false	Preset
	size	300x200	Preset
	title	${system} CPU Queuing	Preset
rpt_system	**template+**	${system} %util% %queue%	Preset
in	list	*	Preset
	recurse	true	Preset
	type	systems	Preset
charts	**over+**	systems	Preset
	sort-order	forward	
	sort-type	name	
rpt_main	**template+**	CPU Utilization and Queuing by System %charts% <hr /> Report Sliding TOC example (version 1.0) Refreshed on $now	Preset

Important notes:

When I have to edit a **template+** field, I always prefer to edit the source rather than the GUI interface. It gives me a much better control on the output than by relying on the wizard. In addition, the wizard often re-formats the text to its own taste and even sometimes discards HTML elements I requested. Also, the wizard is nice but not as much as modern editors. For all these reasons, I edit the code in an external code editor and copy/paste the entire content from my editor to Surveyor's wizard. It's good to know that the standard shortcut keys do work in the editor:

- CTRL-a: select everything, note that you have to click inside the editor panel to select only the text. Otherwise, you'll select the entire page including the Surveyor interface, which is not the purpose.
- DEL: delete the selection.
- CTRL-c: copy the selection to the clipboard.
- CTRL-v: paste the clipboard at the cursor location.

rpt_system is the function that formats the output for a single system. The source reads:

```
00002| <h2 id="${system}">${system}</h2>
00003| %util% %queue%
```

The first line defines a level-2 title. In this case, it's content will be the system name. Some predefined parameters are available to be included in the report.

On the second line, `%util%` insert the CPU Utilization chart and `%queue%` inserts the CPU Queuing chart. This is another advantage of re-labeling the functions: they can be placed exactly where you want rather relying on a one-fits-all default format.

rpt_main is the global glue for the system sub-reports. Its source contents reads:

```
00001| <h1>CPU Utilization and Queuing by System</h1>
00002| %charts%
00003| <hr />
00004| Report <span style="color:#07f;">Sliding TOC example</span> (version 1.0)<br />
00005| Refreshed on $now
```

Line `00001` defines a level-1 title.

Line `00002` places the content of the system sub-reports inside the main output.

Line `00003` places a horizontal line, to separate the report content from the report meta-data.

Line `00004` names the report and version. I advocate storing report meta-data inside the report itself: this way, one can always know where the output comes from, including the software version; in this case, the report itself.

Line `00005` insert a new predefined variable in the meta-data section: `$now` is logically replaced by the current date, which indicates when the report has been generated.

It's now time to save your view once again and once done, open it in a preview and launch it. If everything goes as expected, you report should look like the snapshots on the next page:

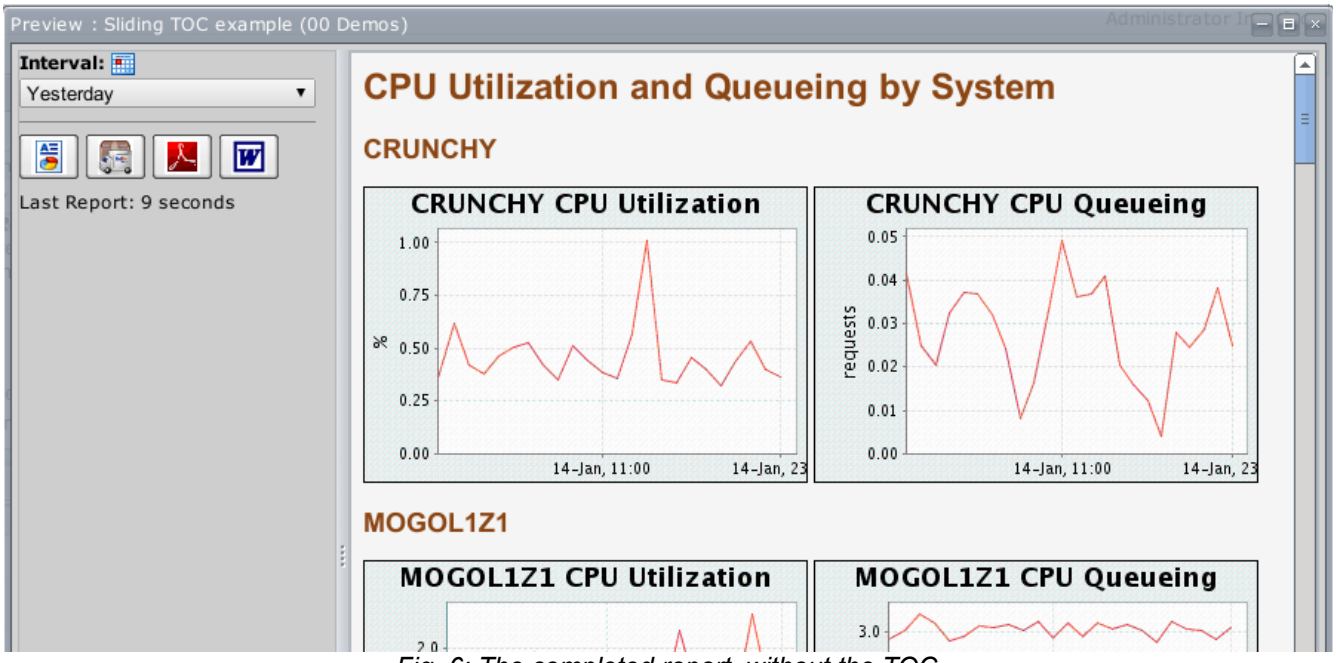

Fig. 6: The completed report, without the TOC

And the bottom section containing the meta-data is as follows:

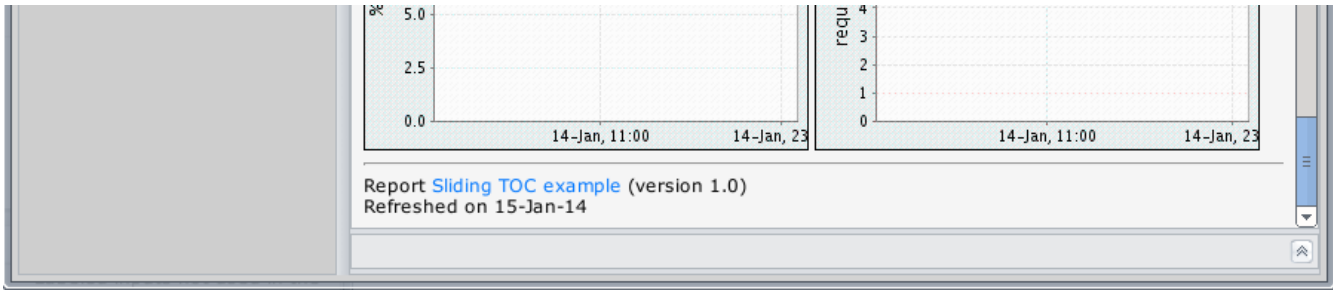

Fig. 7: The report's meta-data section

Enrich the report with the TOC

It is now time to insert the magic stuff in the report. For that, we'll have to update some of the view functions.

Step 1: insert the TOC in the final report

We haven't written the TOC yet (in fact, we did it, but its empty) but we can add in anyway. Update `rpt_main` **template+** according to this (bold blue represents changes against the previous version):

```
00001| <h1>CPU Utilization and Queuing by System</h1>
00002| %rpt_toc% %charts%
00003| <hr />
00004| Report <span style="color:#07f;">Sliding TOC example</span> (version 1.0)<br />
00005| Refreshed on $now
```

This is easy to understand: we just want to insert the TOC in the document.

Step 2: define the variable in var_servers

Update var_servers according to this:

Function label	Property	Value	Input mode
var_servers	**name+**	servers	Preset
	value	`servers = []`	Preset

With this setting, we define a variable named servers (the name property) and assign it to an empty array, denoted by an empty pair of square brackets (the value parameter).

Step 3: feed the servers array

We must now dynamically feed the servers array. For this, we'll use the post-execution field, located on the *common* tab. In this field, enter the following text:

```
00001| servers.push("'${system}'")
```

What's that character soup? 'servers' is the array we want to fill in. '`.push(...)`' is a method that appends a new element to the array. The element appended is contained in the parenthesis. For '`${system}`', we've seen this before while defining `rpt_system`: at execution time, it contains the system name.

It's now time to explain the accolades: in this case, `$system` would do the same job and the accolades are not mandatory. However, if I wanted to push `$system_cpu` instead of `$system`, Surveyor would consider the variable name as being `system_cpu`, which is not defined. By placing the accolades, we notify Surveyor that the variable name is `system` and that its content must be immediately followed by `_cpu`. To be 100% accurate, it's not Surveyor that's obeying but the Groovy shell underneath, Groovy being a programming language. And while talking about Groovy, this also explains this bizarre double set of quotes: we're flying high here: code generating code! Groovy is actually generating a JavaScript statement and therefore, to have an element pushed in the servers array, and given the fact that this `${system}` will be replaced by the effective system name, we must enclose its value in quotes. But the Groovy syntax states that when enclosed in double quotes, the content is interpreted and the quotes are removed from the output. Thus, by keying `servers.push("${system}")`, at execution time, the line generated will in reality be `servers.push(CRUNCHY)` if my server is named `CRUNCHY`. This will in turn be interpreted by the JavaScript as "push the content of the variable named `CRUNCHY`", but `CRUNCHY` is not a variable but a litteral, which explains the presence of the single quotes inside the double ones. Therefore, `servers.push("'${system}'")` is translated in `servers.push('CRUNCHY')`, which is what we wanted. Tricky, isn't it?

Step 4: define the TOC targets

For the TOC to work, we have to place anchors in the report. As we want one anchor per system, the best place is `rpt_system`. We want to update the template+ content as follows:

```
00002| <h2 id="${system}">$system</h2>
00003| %util% %queue%
```

The id parameter acts as anchors in the flow of an HTML page. Later in the step 4, we'll create hyperlinks to link to these anchors.

Note:

If you have system names containing one or more dots, the preceding code must be adapted, since the dot (.) is not allowed in an HTML element's id. Here's the modified version:

```
00001| #set( $cleansrv = $system.toString().replaceAll('[^a-zA-Z0-9]', '-') )
00002| <h2 id="${cleansrv}">${system}</h2>
00003| %util% %queue%
```

Thanks to the Apache Velocity engine embedded in Surveyor, we can create create and use new variables on the fly. That's what this extra line does: `#set` creates a variable named `$cleansrv` and assigns it with the value of

```
$system                 <== this is our good fellow now but...
    .toString()         <== ...it actually is an object and we need a String!
    .replaceAll(        <== A String function to replace every...
      '[^a-zA-Z0-9]',   <== ...character not in the ranges (a to z, A to Z and 0 to 9) ...
      '-')              <== ...with a dash character!
```

Now, on line `00002`, the variable to assign to the id parameter become our new `$cleansrv`. Job done!

Step 5: define the TOC

We're almost done. The last part consists in defining the TOC itself.

I wanted this to be reactive on mouse fly-over movements: whenever the end user flies over the TOC sensitive area, the TOC must unroll from the side. At contrary, when the end user leaves the sensitive area, the TOC must slide to the left and hide.

There are several ways to achieve this. I opted for a simple JavaScript piece of code that uses the jQuery library. No-no, stay with me, don't escape: it's not as frightening as it sounds: within 10 minutes, you'll manipulate that as a pro!

Let's modify the `rpt_toc` **template+** source. Please pay attention to modify the source rather than the GUI, or it won't work. Remember, editing this content in an external code editor makes the job a lot easier!

First, load the jQuery library: if you have our EyeR software package already installed, give yourself a pat in the back for me and enter this line:

```
00001| <script src='/eyer/jquery'></script>
```

EyeR vehicles an instance of jQuery that's easily accessible. If you haven't installed EyeR yet, maybe don't you know there is a free copy download-able from our web site, free copy that includes 10 free licenses for you to try. Anyway, Google provides a safe copy of this library, online, that your browser can access use instantly: there's no install required. Key in the following line, it'll do the same job but the browser needs to be online:

```
00001| <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
```

Define some styling rules

By defining some CSS rules, we'll make the code a bit more readable:

```
00002| <style type="text/css">
00003|   #mytoc {
00004|     position:fixed; top:0; left:-230px; width:260px; height:100%; z-index:1000;
00005|     }
00006|   #mytocInside {
00007|     position:absolute; left:0; top:0; background-color:#E5E5E5;
00008|     border-right: 2px solid #07f; height:100%; width:230px; overflow:auto;
00009|     }
```

```
00010|    #mytocTitle { margin:1em; font-size:1.2em; font-weight:bold; color:#07f; }
00011|    body { margin-left:30px; }
00012|    #hlinks { margin-left:1em; }
00013|  </style>
```

What's remarkable here is the way `#mytoc` is positioned:

- `position:fixed` means that the page scroll won't affect the position of this HTML element
- `top:0` means on top of the page
- `left:-230px` might look weird: we ask to position the TOC outside of the screen, so it'll be hidden, which is what we wanted, after all.
- `width:260px` means the TOC is 260 pixels wide. As per the `left:-230px`, only 30 pixels will be visible on the page: this is our mouse-sensitive area. This portion is present on the page but as the `myTocInside` width is only 230 pixels, the remaining 30 pixels is not visible for the eyes but well for the mouse.
- `height:100%` means the TOC has to cover the whole window height
- `z-index:1000` means that the TOC must appear in the foreground of the other HTML elements.

Define the TOC

Now that the page formatting rules are defined, we can place the elements on the page:

```
00022|  <div id="mytoc">
00023|    <div id="mytocInside">
00024|      <div id="mytocTitle">
00025|        Table of Contents</div>
00026|      <div id="hlinks">
00027|        <div style="font-weight:bold;">
00028|          Systems</div>
00029|      </div>
00030|    </div>
00031|  </div>
```

This is rather obvious. The new element is `<div with id=”hlinks”>`: it contains only a title (Systems), but we'll fill in that in a minute.

Finally, the magic stuff

Complete the script with this code:

```
00032|  <script type='text/javascript'>
00033|    $(document).ready(function() {
00034|      $.each(${servers}, function(){
00035|        var srvclean = this.replace(/[^a-zA-Z0-9]/g, '-')
00036|        var newlink = "<a href='#" +srvclean+ "'>" + this.toString() + "</a><br />"
00041|        $('#hlinks').append(newlink)
00042|        })
```

```
00043|      $('#mytoc').bind('mouseenter', function(){ $(this).animate({'left':'0'}) })
00044|      $('#mytoc').bind('mouseleave',function(){ $(this).animate({
                                                                  ➥ 'left':'-230px'}) })
00050|    })
00051| </script>
```

Notes:

1. The symbol ➥ means “line continuation: this document page is too narrow to display the code as it should.
2. The line numbers are not exactly consecutive but that's on purpose: we'll add functionality in the gaps later on.

Line `00032` tells the browser we're opening a JavaScript script and line `00051` tells the browser the where the script terminates.

Line `00033` is a jQuery shortcut that means “execute this script as soon as the page is completely loaded”. The parenthesis and accolade open on line `00033` are closed on line `00050`.

The line `00034` is also a jQuery syntax to browse through each value stored in the array `${server}`. Does this array remind you something? Yes, absolutely, this is the array we have pushed the system names in the post-execution of our `rpt_system` function. When this will be completed and you will have your view output refreshed, I encourage you to ask your browser to look at the frame source. It's awful code but at the end of the source, you'll find this piece of code and instead of `${servers}` you see... guess what? No guess? Then go look at the source and you'll understand perfectly why you needed to `push(“'${system}'”)` with this strange double quote + single quote syntax; code writing code imposes constraints. The line `00066` closes the parenthesis and accolade open with `$.each(`.

When configuring the `rpt_system` function, I mentioned the fact that a dot is not allowed in a JavaScript id. On line `00035`, you find the dot replacement equivalent. Within this loop, every element of the array will automatically be named `this`.

On line `00036`, we build a new HTML element: `<a href=...>` is a hyperlink. Its simplified form is

```
<a href=”url”>label</a>
```

`label` is what appears on the page

`url` is the address this link must redirect the browser to when one clicks on the hyperlink. In our case, we want an internal link: this is specified by prepending the `url` with a `#` character. Then come the link reference itself, which is the system name without its dots. However, the browser has to display the real host name with its dots and thus, we define the label as being the original system name. The hyperlink is followed by a line break (`<br />`) for a better presentation on the TOC.

To be exact, line `00036` does not build an HTML element but its syntax. The element is effectively created and inserted in the HTML page on line `00041`. The instruction says that the element coded in the variable `newlink` must be appended as child to the element with id `hlinks`.

If you go back to the *Define the TOC* sub-topic, you'll see where these links will appear on the page.

So far, we have a TOC but it's hidden. Hmmm, nice but totally useless! The lines `00043` and `00044` do that job: reveal and hide the TOC. The syntax is rather complex, so let's split it in chunks:

```
00043a|       $('#mytoc')
00043b|                 .bind(
00043c|                       'mouseenter',
00043d|                                    function(){
00043e|                                               $(this)
00043f|                                                      .animate({
00043g|                                                                'left': '0'
00043h|                                                                          })
00043i|                                                                             })
```

Line `00043a` means we operate on the element with id `mytoc`.

On line `00043b`, we apply the method `bind`. The method `bind`, as its name states links an event type with a method to apply when this event occurs. In this case, the event type is `mouseenter`, which means “the mouse enters in the aerial space of this component”. Which component? The one specified on line `00043a`. The syntax on the `00044` is exactly the same but the event type is `mouseleave`, which means “when the mouse leaves the aerial space of this component”.

As the method `bind` links a component, an event type and a function, the line `00043d` initiates this anonymous function.

On line `00043f`, we run the method animate on the component specified on line `00043e`. The animate parameters are given on line `00043g`: set the `left` attribute to `0` (zero). We need a flashback again, when we defined the style rules. At that moment, we stated that the HTML component was to be place 230 pixels left to the display area, and thus invisible. The method `animate` will change this value to `0`, and thus, make it visible. The line `00044` does exactly the same thing, but restores the `left` attribute to its original value: `-230px`, which is “invisible”. The lines `00043h` and `00043i` simply close the open parenthesis and accolades open previously.

Yeeeeehaw! You did it! Test your view, launch it and watch the result!

Oh, by the way: did you notice there wasn't any `%auto-layout%` anymore? It wouldn't be useful, since there's no function underneath the report.

That's nice but we can do much much better than this with a few easy improvements.

Improvement 1: when printing this report, hide the TOC

Very easy, thanks to the CSS media rules. Insert now lines `00042` to `00046`.

```
00013|     @media print { div#mytoc { display: none; } }
```

The CSS rules says that when the media is `print`, the element named `mytoc` may not be displayed.

That's all, improvement in place!

Improvement 2: tell the end user there is a TOC on the page

In fact, if the end user does not fly over the sensitive area with his mouse, he'll never realize there is a TOC. A pity, after all that work done. Let's fix that by adding the lines `00045` and `00046`:

```
00045|          $('#mytoc').animate({ 'left': '0' })
00046|          $('#mytoc').delay(1000).animate({ 'left': '-230px' }, 800)
```

After all new hyperlinks have been created, we want to change the `left` attribute of `mytoc` to 0, to show the TOC. Then, after a delay of `1000` milliseconds, we'll animate `mytoc` again with the attribute `left` restored to its hidden value.

Now test it: when the page loads, the TOC appears for one second, then gently slides left to let the user read the report.

That's all, improvement in place!

A note about the animations: it might seem a waste of time to implement these animations, after all, they look like fancy but useless gadgets, a kind of waste of time. Well, actually, I'm convinced they are not. They refer to our old nature of hunters: the human eyes are automatically attracted by whatever moves in front of us. In addition, the fact that we can see it moving indicates where its hidden and by conclusion, where the user will find it back. Once you have seen it, you'll implicitly know how to use it the next time you face the same situation, which is “this vertical blue line on the left border indicates the is a fold-unfold stuff there”. To “educate” your customers, try to normalize the behavior in your outputs: the customer never read the user manual, utilization has to be instinctive.

Improvement 3: I'm human, I want the servers sorted case-insensitive ('a' must come before 'B')

Easy, replace the line `00034` by this new version:

```
00034|          $.each(${servers}.sort( function(a, b){
                     ➥ return a.toLowerCase().localeCompare(b.toLowerCase()) }), function(){
```

In the previous version, we applied `$.each` on the array `${server}`. Now, before delivering the array, we'll sort it. I won't explain here in details the way `.sort(...)` works, but if you want to know, you find on the internet much more than you'll ever want to know about that.

That's all, improvement in place!

Improvement 4: I want the TOC to fold out when I click on a link

hmmm, this is a bit more complex... We'll change the way we define the links. Replace the line `00036` and add these ones:

```
00036|       var newlink = "<div class='topiclink' " +
00037|                           "onclick=\"$('html, body').animate({ scrollTop: $('#" +
00038|                           srvname + "').offset().top }, 500)\" " +
00039|                           "style='margin-left:3em;'>" + this.toString() +
00040|                       "</div>"
```

Instead of creating `<a...>` elements, we'll simulate them with `<div>` ones.

Next, add the lines `00047` to `00049`:

```
00047|     $('.topiclink').bind('click', function(){
00048|       $('#mytoc').animate({'left':'-230px'})
00049|       })
```

The lines `00047-00049` associate the TOC hiding function to a click on any HTML element of class `topiclink`.

Lastly, define the new links style by adding the lines `00014` to `00020`:

```
00014|   div.topiclink {
00015|     cursor:pointer;
00016|     margin-left:2em;
00017|     color:#07f;
00018|     font-weight:bold;
00019|     }
00020|   div.topiclink:hover { text-decoration: underline; }
```

Test it now, the links are a bit different but the new behavior is implemented.
That's all, improvement in place!

Improvement 5: I don't want to have to copy all this code in every report!

The good news is that Surveyor enables us to define reusable code in a library, so all this code soup does not need to written again and again.

The bad news is that the Surveyor variable function `var_servers` and the code that fills it in (the post-execution code of `rpt_system`) cannot be avoided, but I hope you'll agree, this code portion is minimal.

To create a library, from the main Surveyor menu, select **Edit->Libraries...**, then from the Edit Libraries window menu, select **File->New Library Entry...**. Name the new library, for example, `SlidingTOC`, make sure the Type is set to **template** and press the **OK** button.

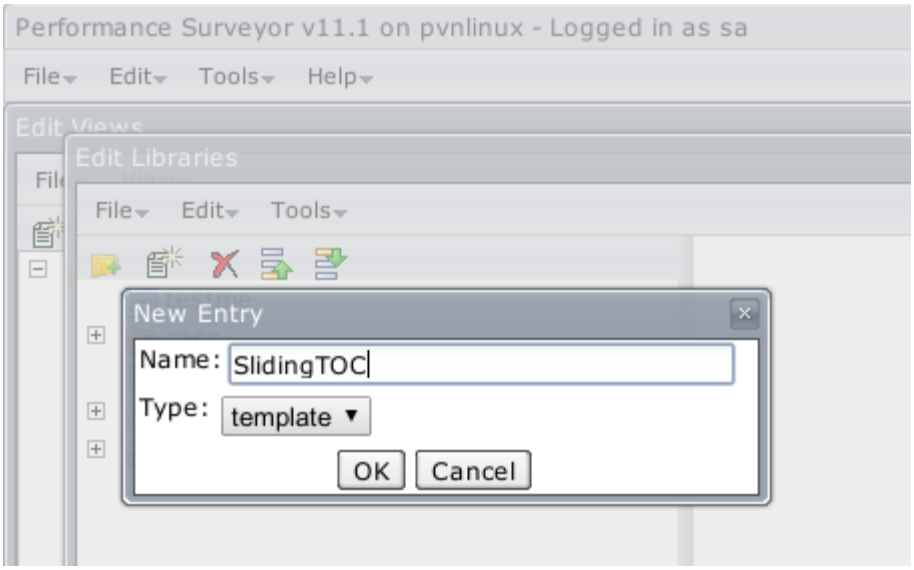

Fig. 8: Create a new library entry

Now, you have to move the whole code from the `rpt_toc` template+ to the new `SlidingTOC` library entry. Lastly, the code in the `rpt_toc` template+ can be removed and the `rpt_toc` library-template attribute can be set to `SlidingTOC` in Preset input mode. You're ready to deploy this TOC in other reports, and you perfectly understand what's happening behind the scene.

Last words

I'll stop here the improvements but there are plenty you can add: for example, instead of just linking to servers, you could want to link to sub-sections of the report: *system xyz, CPU section*, then system *xyz, memory section*. The sky is the limit, be creative: as I said in the introduction, Surveyor is a fantastic tool that's sleeping, waiting for its true power to be revealed.

I also mentioned EyeR in the text. Enable me to introduce this new beast in a few words: EyeR is a primarily a set of views dedicated to target specific issues. These views work out-of the-box but are also widely customizable. As we believe in our solution, we made it freely available for download from our web site (http://www.creative-associates.be). Install for free, play with the 10 welcome free licenses, experiment and if you like it, be aware that additional licenses cost a hundredth, yes, 1/100 the price of a Surveyor license.

Should you have a question about something not clear in this document, or have a comment, discover a mistake, or want to suggest an improvement, please do not hesitate to send me a quick email at info@creative-associates.be. Unconditional enthusiasts who would like to send some chocolates, you'll find our postal address on our web site ;-)

Maybe one day, you'll see people playing like kids with the TOC. Just keep smiling, you know now how magic is easy once you know the trick!

Thanks for reading this so far and if you really completed this tutorial, please give yourself another pat in the back for me: you deserve it, at least for being that indulgent with my poor nerd prose. Cheers.