Table of Contents

Polymer, SWATCH and the

Nomenclature

Legacy (TriggerSupervisor) = Trigger Supervisor as used for the trigger prior to the 2016 upgrade

Polymerized (TriggerSupervisor) = Trigger Supervisor as used for the trigger for the 2016 upgrade

Polymer Element = Similar to a Dojo widget but more customizable

Polymer in a nutshell

Polymer is a framework developed by Google that eases the definition of so-called shadow-DOMs. That's fancy words for "it allows you to define your own HTML tags". The intricacies of how that works are rather complex and interesting, so we encourage you to read this (link to original Glenn TWiki).

Such a shadow DOM (or custom HTML tag) is called an Element in Polymer. An element defines a "sort-of" namespace for CSS-styling and javascript. From the HTML view-point elements can be considered like any other HTML tag. From the Trigger Supervisor view-point they can be considered as Objects in the C++ sense: They can have properties (members), member-functions, etc.

Data Binding (read at least this)

The nicest thing about Polymer elements is that they can have properties. These properties are exposed to HTML (and therefore your TS C++ code) via HTML attributes. Properties can be any native javascript type (String, Boolean, ...) or even objects and arrays in JSON syntax.

Polymer in the Trigger-Supervisor context:

The main change (and good news) is that anything to do with HTML now lives in `your-system/your-cell/html`. (Which already existed but was barely used.) A polymer element lives in `your-cell/html/elements/your-element/`

The remaining code structure remains the same as in the legacy.

Porting legacy to Polymer

To first order, all you need to do is to compile your pre-existing Cell against the new Polymer TS release. Of course your cell will continue Dojo widgets if you don't port the corresponding parts. More on that later.

Install the Polymer Beta Release of the Trigger Supervisor:

```
Add how to install the Polymer release!
```

Now make sure to recompile your Cell and panels:

```
make clean && make && make rpm && make install
```

Porting legacy Dojo widget X to Polymer element Y

For many of the typical Dojo widgets (like input forms, buttons etc) pre-defined Polymer Elements exist. All elements in the standard Polymer Element catalogue are installed.

E.g. for input forms, look at the paper elements. These are included by default when compiling against the new RPMs. However, to enable callbacks to C++ on submission of any form, you'll have to write your own custom polymer element (that make use of the pre-defined Polymer elements).

Writing a custom Polymer Element

A polymer element consists of up to three files, one for the HTML code, one for SCSS and one for javascript.

Example

Let's assume you're building an element called `your-element` that lives in `your-cell/html/elements/your-element`:

Code in `your-cell/html/elements/your-element/your-element.html`

```
<dom-module id="your-element">
    <template>
        <link rel="import" type="css" href="css/your-element-min.css">
        <h1>{{headline}}</h1>
        <div class="container">
            <content></content>
        </div>
    </template>
    <script src="javascript/your-element-min.js"></script>
</dom-module>
```

Code in `your-cell/html/elements/your-element/css/your-element.scss`

```
h1 {
  font-size: 20px;
  color: gray;
}
.content {
  font-size: 12px;
}
```

Code in `your-cell/html/elements/your-element/javascript/your-element.js`:

```
Polymer({
    is: "your-element",
    properties: {
        headline: {
            type: String,
            value: 0
        }
   }
});
```

"Building" Polymer elements

In order to use a custom Polymer element, the TS framework has to know about it. All elements are included in the HTML when the ajaxell page is built. The framework searches each Cell's html folder for a file `elements.html` which "defines" all custom Polymer elements in that cell. In our example above this would look like:

Code in `your-cell/html/elements/elements.html`:

```
<link rel="import" href="./your-element/your-element.html">
```

You may have noticed that above you define `your-element.js=/=scss` while in `your-element.html` you include `your-element-min.js=/=css`. These files are generated by a build system from the files you actually write (SCSS to CSS and js to min.js). One benefit of this procedure is that the file-sizes are reduced by removing unnecessary whitespace for the files loaded by the browser while having well-formed code in the files which you develop. This is done by a tool called grunt.

Installing grunt and pre-requesites

Grunt needs Ruby we recommend installing RVM (a Ruby Version Manager) and SASS (a Ruby package installed with gem). Grunt is installed with NPM (Node Package Manager a widespread javascript library package manager) so you need to install those two, too:

```
#in case you don't have ruby
gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3
\curl -sSL https://get.rvm.io | bash -s stable --ruby

source [your-user-home]/.rvm/scripts/rvm

#Sass
gem install sass

#npm
sudo yum install npm

#grunt
npm install -g grunt-cli
```

Actually building your Polymer element

Now you'll have to define your Grunt project. Best you start from the package and gruntfile in the ajaxell folder:

```
# in your-cell/
svn export svn+ssh://svn.cern.ch/reps/cactus/trunk/cactuscore/ts/ajaxell/package.json
npm init
cd html
svn export svn+ssh://svn.cern.ch/reps/cactus/trunk/cactuscore/ts/ajaxell/html/gruntfile.js
ln -s ../node_modules node_modules
grunt
```

Adding the element to your Panel / Cell

To tell Ajaxell that your elements exist you have to add in the `Cell.cc` in `Cell::init()`:

```
void
Cell::init() {
  #ifdef POLYMERIZE
    getContext()->getCell()->getApplicationDescriptor()->setAttribute("appPath", "your-system/you
  #endif
}
```

Depending on the use-case you may skip the if clause. In your panel you now have access to any elements defined in `your-cell/html/elements/elements.html`. Add the polymer element with:

```
ajax::PolymerElement* myElement =  new ajax::PolymerElement("your-element");
myElement->set("headline", "Headline");
myElement->add((new ajax::PlainHtml("Lorem ipsum sic amet."));
panel->add(myElement);
```

Resulting HTML:

```
<your-element headline="Headline">Lorem ipsum sic amet</your-element>
```

Equivalent HTML (or what is actually rendered):

```
<h1>Headline</h1>
<div class="content">
  <content>Lorem ipsum sic amet</content>
</div>
```

Callbacks to C++

SWATCH specifics

Running your cell standalone

Pre-defined elements

Pre-defined styles

-- JoschkaLingemann - 2015-09-28

This topic: Sandbox > JoschkaLingemannSwatch
Topic revision: r3 - 2015-10-08 - CarlosGhabrous