How to use the structurer

This How-To describes the usage of the structurer config domain specific language to create beautiful websites in no time.

Table of contents

1. Intended Audience

Warning:

The "Dispatcher" (aka "Views") is new functionality which is still in development phase. That is why it is in the "whiteboard" section of the Forrest distribution. This HowTo is a good start but still needs more work.

This part of the the dispatcher is called the structurer and is dedicated to webdesigner and user with some knowlegde of css.

Note:

For the moment we will use a special seed template called seed-v2. We still need to fix issues of the dispatcher that it really can replace old fashion skins. Like performance, standalone contracts, other testing output/input formats - using POJO based processing will help solving this. All this will ATM happen in the v2 seed-target till the dispatcher will be realesed to the stable plugins. So make regular updates of your forrest-trunk to keep track.

Warning:

The way we develop contracts will/may change with introduction of java based processing rather then xsl. Please keep this in mind and help updating the documentation by sending patches. TIA. ;-)

2. Purpose

This how-to will show you how to write a **forrest:view** from the ground up. We will focus on html as the output format. As well it will show how to add your own css implementation to the structurer.

3. Prerequisites

- You have a ready-to-go new seed-v2 (v2) based on the dispatcher like described in Install.
- Reading that how-to is as well a good idea to understand the used directory structure in this how-to.
- Installing a mozilla browser and the forrestbar helps a lot in developing.

4. Steps

Note:

When developing with the dispatcher we assume you are using 'forrest run' and the following workflow "change files -> refresh browser" Installing a mozilla browser and the forrestbar helps a lot in developing. Many instructions assumes that you have the forrestbar installed.

We developed **the structurer** to let the user decide where to place elements in e.g. html pages. We started this work with the `skinconf.xml` where you could configure certain elements and their positions. These elements were known under certain names. It was up to the skin designer to support this configuration and the elements.

The work started with grouping elements (the ones from skinconf). We used css-contracts that we added as @attributes e.g. `<div id="content-main"/>` . That made it possible to use the same elements in different skins. For the full list refer to the initial contract list

Around this contracts we developed a configuration Domain Specific Language - called **the**

structurer. The **structurer** allows us to define the order in which **forrest:contract**s appear, and also to group them using **forrest:hook**s.

forrest:hooks are containers that are only used for layout reasons. They **do not** add any content nor functionality to the output. They add **only** layout information to the output. Actually e.g. a `<forrest:hook name="layoutId"/>` will be transformed to `<div id="layoutId"/>`

forrest:contracts are functionality or extra content that a theme can use to display the request. Sometimes a contract delivers **format-specific markup**, other times it delivers a **format-independent string**. We decide different kind of contracts, static one (like described in the contract howto), semi static (which offer configuration parameter in the structurer) and dynamic contracts (which offer semi-static configuration and/or requesting the content).

Till now the processing includes firstly all raw data into the structurer, then generating a dynamic xsl and last but nor least response. That is heavy based on xsl processing which is quite slow. A better way is that we do not include *raw* but the *transformed* (by the contract) data . This way we do not have to generate a dynamic stylesheet which leads to the upcoming development of standalone contracts.

Note:

The structurer is as well a configuration file for the dispatcher. The new think on the dispatcher is that one can include any content from any given business service by dispatching a request against it. In "old fashion" skins and in v1 contracts we assumed a given data model. In the dispatcher there is **no** given data model any more. All data has to be defined in the structurer that they can be dispatched.

4.1. Empty structurer file

```
<forrest:views
  xmlns:forrest="http://apache.org/forrest/templates/1.0"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <forrest:view type="html">
  </forrest:view>
</forrest:views>
```

The structurer is designed to be open for any format that can use **forrest:view** as configuration file. The only format we implemented is html for now. This is as well true for the delivered contracts.

4.2. Creating your first structurer

Warning:

The structurer is based on jx templates to allow simple presentation logic (all code starting with "jx:"). Please refer to the cocoon documentation about jx. For now we are using jx to include the raw data into the presentation model and generating an alias-xsl stylesheet. That is heavy on performance and we will change this ASAP. Mind the warning at the start of the howto.

In this section we will create a new structurer. We will override the default structurer of the themer-plugin for the index page of the v2. For that we will create a file called `index.fv` and save it in our xdocs directory. This will make **only** the index.html page look different from the rest of the project.

Note:

You can set a view for an individual file, a directory, or the whole site. To address multiple files in a directory call your `.fv` file `common.fv`. If Forrest doesn't find a `.fv` file with the same name as the current file it will use the common.fv file in that directory, or the first one it finds going upwards through the directory structure. `common.fv` files affect all subdirectories unless they are overidden by another `common.fv` or a file-specific `foo.fv` file.

Remember: pointing your browser to `http://localhost:8888/ls.contracts.html` will

show a page with all contracts and themes that you can use in your project provided by forrest.

Let us use the blank structurer from the earlier step and add the content-main contract. In ls.contracts.html we find the information for how to use the contract in our structurer. Our `index.fv` should look like:

```
<forrest:views
  xmlns:forrest="http://apache.org/forrest/templates/1.0"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <forrest:view type="html">
    <forrest:contract name="content-main">
    <forrest:properties contract="content-main">
      <forrest:property name="content-main" nugget="get.body">
        <jx:import uri="cocoon://#{$cocoon/parameters/getRequest}.body.xml"/>
      </forrest:property>
      <!-- Heading types can be clean|underlined|boxed  -->
      <forrest:property name="content-main-conf">
        <headings type="underlined"/>
      </forrest:property>
    </forrest:properties>
  </forrest:contract>
  </forrest:view>
</forrest:views>
```

A contract has to request the data model it want to transform. This happends by defining forrest:properties which have the same name like the contract. In our case we want the HTML rendered from intermediate format (**.body.xml). This we are going to include via: `<jx:import uri="cocoon://#{$cocoon/parameters/getRequest}.body.xml"/>`

Contracts can offer some property configuration of the outcome of the transformation. In our case `<forrest:property name="content-main-conf"> <headings type="underlined"/> </forrest:property>.`

Lets try our new structurer by pointing to `http://localhost:8888/index.html`. We will see only the main content. Now let us add the section navigation to our structurer. The contract usage in the structurer can be looked up in ls.contracts.html. Our structurer now looks like:

```
<forrest:views
  xmlns:forrest="http://apache.org/forrest/templates/1.0"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <forrest:view type="html">
    <forrest:contract name="nav-main">
      <forrest:properties contract="nav-main">
        <forrest:property name="nav-main" nugget="get.navigation">
          <jx:import
            uri="cocoon://#{$cocoon/parameters/getRequest}.navigation.xml"/>
        </forrest:property>
      </forrest:properties>
    </forrest:contract>
    <forrest:contract name="content-main">
      <forrest:properties contract="content-main">
        <forrest:property name="content-main" nugget="get.body">
          <jx:import uri="cocoon://#{$cocoon/parameters/getRequest}.body.xml"/>
        </forrest:property>
        <!-- Heading types can be clean|underlined|boxed  -->
        <forrest:property name="content-main-conf">
          <headings type="underlined"/>
        </forrest:property>
      </forrest:properties>
    </forrest:contract>
  </forrest:view>
</forrest:views>
```

We now find the main content and the section navigation after each other and in the order we placed

them in the structurer, but we want it next to each other (left: nav-section; right: content-main).

4.3. Hooks in the structurer

We will use now the first time a `<forrest:hook name="layoutId"/>` . Hooks are the styling side of the structurer. We can imitate arbitrary html skeleton with their help. Before we explain how to use your own css in the structurer, we will use the default css. You can see in our example that we have css included. That is a default fallback coming from the implementation. In this common.css we can find

```
/* menu */
#leftbar {
    width: 25%;
    float: left;
    background: #eae8e3;
    border: thin dashed #565248;
}
```

With this information we know to use `<forrest:hook name="leftbar"/>` and add contracts into that container.

If we want to put the nav-section contract into the left-hand side position of the site we need to place the contract into that hook. Like:

```
<forrest:hook name="leftbar">
  <!-- Include contract here -->
</forrest:hook>
```

Our structurer will then look like:

```
<forrest:views
  xmlns:forrest="http://apache.org/forrest/templates/1.0"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <forrest:view type="html">
    <forrest:hook name="leftbar">
      <forrest:contract name="nav-section">
        <forrest:properties contract="nav-section">
          <forrest:property name="nav-section" nugget="get.navigation">
            <jx:import
              uri="cocoon://#{$cocoon/parameters/getRequest}.navigation.xml"/>
          </forrest:property>
        </forrest:properties>
      </forrest:contract>
    </forrest:hook>
    <forrest:contract name="content-main">
      <forrest:properties contract="content-main">
        <forrest:property name="content-main" nugget="get.body">
          <jx:import uri="cocoon://#{$cocoon/parameters/getRequest}.body.xml"/>
        </forrest:property>
        <!-- Heading types can be clean|underlined|boxed  -->
        <forrest:property name="content-main-conf">
          <headings type="underlined"/>
        </forrest:property>
      </forrest:properties>
    </forrest:contract>
  </forrest:view>
</forrest:views>
```

4.4. CSS in the structurer

We now know how to place contracts and hooks in our structurer. Until this stage we only used the

common.css. CSS-support of the structurer is as easy as placing contracts/hooks. To override the common.css stylesheet we use another tag within our structurer `<forrest:css />` .

You can add inline and linked css with the structurer. As soon as you use forrest:css you will disable the fallback css support from forrest. With this tag we tell the dispatcher that we want to override the common.css. After adding the following to our index.fv the design will be different.

```
<forrest:css >
/* Extra css */
/* menu */
#leftbar {
  width: 25%;
  float: left;
  background: #CCCCFF;
  border: thin solid #000000;
}
</forrest:css>
```

We just changed the border-style to 'solid', the background to '#CCCCFF' and the color to '#000000'. So you see a white page where the menu is surrounded by a solid border with the defined background.

Note:

`<forrest:css />` needs to be the direct child of `<forrest:view type="html">`

```
<forrest:views
  xmlns:forrest="http://apache.org/forrest/templates/1.0"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <forrest:view type="html">
    <forrest:css >
/* Extra css */
/* menu */
#leftbar {
  width: 25%;
  float: left;
  background: #CCCCFF;
  border: thin solid #000000;
}
    </forrest:css>
    <forrest:hook name="leftbar">
      <forrest:contract name="nav-section">
        <forrest:properties contract="nav-section">
          <forrest:property name="nav-section" nugget="get.navigation">
            <jx:import
              uri="cocoon://#{$cocoon/parameters/getRequest}.navigation.xml"/>
          </forrest:property>
        </forrest:properties>
      </forrest:contract>
    </forrest:hook>
    <forrest:contract name="content-main">
      <forrest:properties contract="content-main">
        <forrest:property name="content-main" nugget="get.body">
          <jx:import uri="cocoon://#{$cocoon/parameters/getRequest}.body.xml"/>
        </forrest:property>
        <!-- Heading types can be clean|underlined|boxed  -->
        <forrest:property name="content-main-conf">
          <headings type="underlined"/>
        </forrest:property>
      </forrest:properties>
    </forrest:contract>
  </forrest:view>
</forrest:views>
```

As a second example, let us change as well the content-main by adding another hook `<forrest:hook name="content"/>` We need to add the new layout container to our inline

css:

```
/* The actual content */
#content {
  margin-left: 25%;
  padding: 0 20px 0 20px;
  background: #B9D3EE;
}
```

Then we have to add the 'content-main' contract to the 'content' hook. The resulting structurer looks like:

```
<forrest:views
  xmlns:forrest="http://apache.org/forrest/templates/1.0"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <forrest:view type="html">
    <forrest:css >
/* Extra css */
/* menu */
#leftbar {
  width: 25%;
  float: left;
  background: #CCCCFF;
  border: thin solid #000000;
}
/* The actual content */
#content {
  margin-left: 25%;
  padding: 0 20px 0 20px;
  background: #B9D3EE;
}
    </forrest:css>
    <forrest:hook name="leftbar">
      <forrest:contract name="nav-section">
        <forrest:properties contract="nav-section">
          <forrest:property name="nav-section" nugget="get.navigation">
            <jx:import
              uri="cocoon://#{$cocoon/parameters/getRequest}.navigation.xml"/>
          </forrest:property>
        </forrest:properties>
      </forrest:contract>
    </forrest:hook>
    <forrest:hook name="content">
      <forrest:contract name="content-main">
      <forrest:properties contract="content-main">
        <forrest:property name="content-main" nugget="get.body">
          <jx:import uri="cocoon://#{$cocoon/parameters/getRequest}.body.xml"/>
        </forrest:property>
        <!-- Heading types can be clean|underlined|boxed  -->
        <forrest:property name="content-main-conf">
          <headings type="underlined"/>
        </forrest:property>
      </forrest:properties>
    </forrest:contract>
    </forrest:hook>
  </forrest:view>
</forrest:views>
```

We are now able to place contracts into the layout container and add custom css to the structurer.

FIXME (thorsten):

Add more information of recent threads around css in the structurer

5. Further Reading

Congratulations you are now able to work with the structurer. From here we recommend to read the following How-Tos:

- Create your own contract implementation

6. Feedback

Please provide feedback about this document via the mailing lists.