Using the HTML Element Spawner

The XNAT Spawner is a way to generate UI elements on-the-fly with JavaScript and simple configuration objects (in YAML, JSON, or JavaScript). Minimal configuration is needed to render UI elements and form controls that are structured and styled for easy integration with XNAT, and in many cases JavaScript will not need to be written at all (more complex elements may need custom widget functions).

There's a back-end XAPI Spawner Service that can parse YAML files, resolve references in the YAML files, and generate JSON that the front-end Spawner JavaScript functions use as input to recursively generate nested HTML DOM elements.

YAML Files

When defining your Spawner elements using YAML, XNAT can automatically resolve elements using an XAPI REST service from files placed in specific places. The XAPI Spawner service translates YAML into JSON which is consumed by the Spawner JS engine to create and render UI elements.

Files placed in the **`src/main/resources/META-INF/xnat/spawner`** directory will be parsed by the Spawner service when XNAT starts. A namespace will be automatically created based on the file name - it will convert the **hyphenated** file name to the **camelCase** equivalent (and drop the **-elements.yaml** suffix, if present) and calls to **`/xnat/spawner/resolve/nameSpace/elementName`** will be resolved accordingly.

For example, for a file named **`sample.yaml`** (in the directory mentioned above), containing the following YAML code…

sample-elements.yaml

```
root:
    kind: tabs
    label: Foo Tabs
    container: div.xnat-tab-container
    contents:
        bar:
            kind: tab
            label: Bar Tab
            active: true
            contents:
                ${userInfo}

userInfo:
    kind: panel.form
    label: User Information
    url: /xapi/users
    method: POST
    contentType: json
    contents:
        ${username}
        ${email}

username:
    kind: panel.input.text
    name: username
    label: Username
    placeholder: username

email:
    kind: panel.input.email
    name: emailAddress
    label: Email Address
    placeholder: user@domain.org
```

…a **`GET`** request to **`/xapi/spawner/resolve/sample/root`** will return the following JSON…

/xapi/spawner/resolve/sample/root

```
{
   "root": {
       "kind": "tabs",
       "label": "Foo Tabs",
       "container": "div.xnat-tab-container",
       "contents": {
           "bar": {
               "kind": "tab",
               "label": "Bar Tab",
               "active": true,
               "contents": {
                   "userInfo": {
                       "kind": "panel.form",
                       "label": "User Information",
                       "url": "/xapi/users",
                       "method": "POST",
                       "contentType": "json",
                       "contents": {
                           "username": {
                               "kind": "panel.input.text",
                               "name": "username",
                               "label": "Username",
                               "placeholder": "username"
                           },
                           "email": {
                               "kind": "panel.input.email",
                               "name": "emailAddress",
                               "label": "Email Address",
                               "placeholder": "user@domain.org"
                           }
                       }
                   }
               }
           }
       }
   }
}
```

…and when processed with the Spawner JS engine…

XNAT.spawner.resolve

```
// resolves url: /xapi/spawner/resolve/sample/root -- only namespace and element are used here
// a selector or DOM element is passed as the argument to the .render() method
XNAT.spawner.resolve('sample/root').render('div.xnat-tab-container');
```

…the following HTML is produced…

HTML Output

```
<div class="content-tabs xnat-tab-container" style="display: block;">
    <div class="xnat-nav-tabs side pull-left">
        <ul class="tab-group" style="display: block;">
            <li class="tab active" data-tab="bar"><a title="Bar Tab" href="#bar">Bar Tab</a></li>
        </ul>
    </div>
    <div class="xnat-tab-content side pull-right">
        <div class="tab-pane active" data-name="bar" data-tab="bar">
            <form class="xnat-form-panel panel panel-default" id="user-info-panel" name="userInfo" action="/xapi
/users" data-panel="user-info" data-method="post" method="POST" data-status="clean">
                <div class="panel-heading"><h3 class="panel-title">User Information</h3></div>
                <div class="panel-body">
                    <div data-name="username" class="panel-element">
                        <label class="element-label" for="username">Username</label>
                        <div class="element-wrapper">
                            <input type="text" id="username" name="username" size="25" title="Username"
placeholder="username" data-value="">
                            <div class="description"></div>
                        </div>
                        <br class="clear">
                    </div>
                    <div data-name="emailAddress" class="input-text email panel-element">
                        <label class="element-label" for="email-address">Email Address</label>
                        <div class="element-wrapper">
                            <input type="text" class="input-text email" id="email-address" name="emailAddress"
size="25" title="Email Address" placeholder="user@domain.org" data-value="">
                            <div class="description"></div>
                        </div>
                        <br class="clear">
                    </div>
                </div>
                <div class="panel-footer">
                    <button class="submit save pull-right btn btn-sm submit" type="submit">Save</button>
                    <span class="pull-right">   </span>
                    <button class="revert pull-right btn btn-sm" type="button">Discard Changes</button>
                    <div class="clear"></div>
                </div>
            </form>
        </div>
    </div>
</div>
```

For the **`tabs`** widget, the outer **`container`** element needs to exist on the page first - the tabs need a target to spawn into.

For a working example, examine the YAML file used to render the Site Administration page:

Site Administration YAML File

```
src/main/resources/META-INF/xnat/spawner/site-admin.yaml
```

Sample YAML Code

Spawner Text Input Element

The configuration below is a pre-defined **`panel.input.text`** widget that generates a text input with the proper HTML structure and styling for use in a form panel.

Config Object

```
siteId:
   kind: panel.input.text
   name: siteId
   label: Site ID
   validation: required id onblur
   description: >
       The id used to refer to this site (also used to generate
       database ids). The Site ID must start with a letter and
       contain only letters, numbers and underscores. It should
       be a short, one-word name or acronym which describes your
       site. No spaces or non-alphanumeric characters.
```

Element config objects may share some common properties like **`kind`** (which is required for predefined element types) and **`name`**, but other properties may differ depending on the element type.

Shown below is the HTML that's generated when the YAML code above is parsed and processed by the Spawner service and JS engine. It contains HTML elements with the necessary attributes for proper styling and functionality.

HTML Output

```
<div data-name="siteId" class="panel-element">
   <label class="element-label" for="site-id">Site ID</label>
   <div class="element-wrapper">
       <input class="required id onblur" type="text" id="site-id" name="siteId" size="25" title="Site ID" data-
value="" data-validate="required id onblur">
       <div class="description">The id used to refer to this site (also used to generate database ids). The
Site ID must start with a letter and contain only letters, numbers and underscores. It should be a short, one-
word name or acronym which describes your site. No spaces or non-alphanumeric characters.</div>
   </div>
</div>
```

"Where'd the **`id`** on the <input> element come from? It's not specified in the config object!"

For some elements, if no **`id`** attribute is explicitly defined, a hyphenated version of the element's **`name`** will be used. Not all elements require ids, though, so they're not always generated.

Spawner Freeform Element

You can generate pretty much any HTML element using the Spawner, just enter the tag name in the **`tag`** property and the attributes and properties for that element in the **`element`** property. HTML content that needs to be inserted into the element can be entered in a **`content`** property at the root of the config object or as an **`html`** property on the **`element`** property.

Config Object

```
siteDescriptionText:
    tag: textarea
    element:
        name: siteDescriptionText
        rows: 8
    after: "<p>Specify a simple text description of this site.</p>"
```

The **`after`** property can accept either an HTML string or a child Spawner object and will be inserted after the generated element.

Here's the HTML output from the object above:

HTML Output

```
<textarea id="site-description-text" name="siteDescriptionText" rows="8"></textarea>
<p>Specify a simple text description of this site.</p>
```

Spawner Element Variable Resolution

You can also reference other Spawner elements by name using a syntax - `${variable}` - similar to BASH or JSP. The example below defines an element which is referenced in the element below it. Referencing other elements like this is a good way to keep the YAML organized and easy-to-read as you nest elements for more complex layouts.

```
adminEmail:
    kind: panel.input.email
    id: admin-email
    name: adminEmail
    label: Site Admin Email
    description: >
        The administrative email account to receive system emails. This
        address will receive frequent emails on system events, such as
        errors, processing completion, new user registration and so on.
        These emails can be configured on the Notifications tab.

adminInfo:
    kind: panel.form
    name: adminInfo
    label: Admin Information
    method: POST
    contentType: json
    url: /xapi/siteConfig
    contents:
        ${adminEmail}
```

The **`${adminEmail}`** variable is resolved with the XAPI Spawner Service and inserts the **`adminEmail`** element that is defined above at the variable location.

(The form properties used here are described in the XNAT Spawner Elements documents.)

Spawner Widget Methods

The **`kind`** property maps to the JavaScript method used to render the widget. In the example at the top of the page (the **`siteId`** element), the method name happens to be in the **`XNAT.ui`** namespace, so the **`XNAT.ui.panel.input.text()`** method is called using the properties in the YAML config object. If no function exists at **`XNAT.ui.panel.input.text`**, the namespace hierarchy will be searched up to the global (window) namespace for a matching method name. If no matching method exists, no element will be generated.

Here's the order for which widget methods are searched:

```
XNAT.kind.init()
XNAT.kind()
XNAT.ui.kind.init()
XNAT.ui.kind()
kind.init()
kind()
```