Contents

Building GWT

The GWT widgets are built with opentaps when using the default ant task.

To speed up compilation you can build the GWT widgets independently of opentaps:

```
$ ant gwt
```

To clear the previous gwt build:

```
$ ant clean-gwt
```

Or combine the two steps:

```
$ ant refresh-gwt
```

Note that it is also possible to build only the server code (skipping the GWT widgets), for this you can use the following commands:

```
$ ant server
$ ant clean-server
$ ant refresh-server
```

Building the GWT widgets will cause ant to look for look "gwt" in the opentaps components' build.xml files and build them one at a time. In the component build.xml, the following directories are specified for building gwt:

```
<property name="gwt.deploy.dir" value="./webapp/crmsfagwt"/>
<property name="gwt.module.base" value="org.opentaps.gwt.crmsfa"/>
<property name="gwt.src.common" value="../opentaps-common/src/org/opentaps/gwt"/>
<property name="gwt.src.base" value="./src/org/opentaps/gwt/crmsfa"/>
```

Then, when ant tries to build gwt, it will look all that gwt modules specified in the build.xml. Each module is specified at a path of `${gwt.deploy.dir}/${gwt.module.base}.${module}.${module}` For example, if you specify `contacts` as the module to compile, then opentaps will try to compile `org.opentaps.gwt.crmsfa.contacts.contacts.gwt.xml`, which should be in your src/ path.

When you have an additional GWT module to build, add it to the list of modules:

```
<foreach list="contacts,accounts,leads,partners" target="gwtcompile" param="module"/>
```

To speed up the build during development, you can setup GWT to only compile for one of the supported browsers. This is configured in the common module in hot-deploy/opentaps-common/src/org/opentaps/gwt/common/common.gwt.xml. For example, you can enable it for only Mozilla/Firefox by setting the user.agent property to "gecko1_8":

```
<set-property name="user.agent" value="gecko1_8"/>
```

Configuring Server Side Interaction

Your GWT widgets will need to interact with server-side services to store and retrieve data. A "best practices" pattern we have started in opentaps is to create a configuration Java file for each server side service available for GWT client-side widgets. For example, there is a `org.opentaps.gwt.crmsfa.contacts.client.form.configuration.QuickNewContactConfi` Class which contains the server-side URL and all the form parameters for interacting with the quick new contact service on the server. This is part of the GWT client package and is designed to be used by all the client-side widgets. Note that the pattern is to have one Configuration Java file for each *server*-side service, to be shared by many client-side widgets which may access the same server-side service, not to have a configuration file for each client-side widget.

Permission

The GWT widgets do not perform security checks, but users permissions are made available to them in order to adapt the user interface accordingly. The real security checks allowing a user to perform an operation or retrieve data are performed server side.

Client-side permission checking is handled in the following way:

1. The server side uses the `User` object to determine what permissions the currently logged in user has and puts it into the webpage sent to the client as an object using JavaScript. This is done in the `main-decorator.bsh` and `header.ftl` of the server.
2. On the client-side, the `Permission` class retrieves the security permissions set into the browser via JavaScript. Your GWT widget can use its `hasPermission` method to check if the user has permissions to access certain sections of your page:

```
if (!Permission.hasPermission(Permission.CRMSFA_CONTACT_CREATE)) {
    return;
}
```

WARNING: Do not rely on those checks to hide sensitive data or services from a user. Specifically, it is possible for the end user to modify the JavaScript and add permissions for their displayed widgets. Therefore,

you should always filter out sensitive data before sending them to the client-side widgets, and every operation on the server side should check permissions again. Client-side widget permission checking is only for hiding parts of the user interface and should not be considered a security feature.

Images and CSS

Images and CSS for the GWT widgets are in the `org/opentaps/gwt/common/public` directory under `hot-deploy/opentaps-common/src/common/`

Note that sometimes you have to use `!important` for your CSS styles, as your style may be inheriting from other styles.

Building Widgets with Base Panels

Base panels are base classes providing handlers and utility methods to quickly build forms that integrate with the application.

The Base Classes

BaseFormPanel is the base of all forms and it provides the following:

- **addField** and **addRequiredField** methods that set the correct class for the labels and set the handler that submits the form when the user hits the Enter key
- **addStandardSubmitButton** that places a submit button
- the three **FormListener** event handlers that performs validation before submitting, display an activity indicator when the form is submitted, handles exception returned by the server
- a mechanism to notify any registered widgets when the form has been successfully submitted

ScreenletFormPanel is the base class for forms that should fit on the left column, in all aspects it behaves the same as **BaseFormPanel** only the CSS classes applied differ.

TabbedFormPanel provides methods to create a multiple tabs form such as the one used to present the filters available in Find Contacts. The tabs created are **SubFormPanel** which provide the same add fields methods than **BaseFormPanel**.

Validation

BaseFormPanel provides simple validation that is automatically called before trying to submit the form. It works by checking each field in the form against its own internal validation method, for example it checks that all required fields are filled, that email address input fields have valid email addresses, etc ...

In order to implement a more complex validation, simply override the *isValid()* method (be sure to call the base implementation first to keep checking field validation methods).

Notification

In order to notify another widget it must first implement the **FormNotificationInterface**. You can register (using the *register()* method) that widget in your form and the method *notify()* of that widget will get called if your form was successfully submitted. This is normally done in the entry class which is the place where all widgets are loaded.

For example, the **PartyListView** implements it allowing form that create new parties like the **QuickNewContactForm** to notify the list that is should reload in order to display the newly created entity.

And the notification is setup in the contact entry point:

```
if (RootPanel.get(QUICK_CREATE_CONTACT_ID) != null) {
    loadQuickNewContact();
    // for handling refresh of lists on contact creation
    if (myContactsForm != null) {
        quickNewContactForm.register(myContactsForm.getListView());
    }
    if (findContactsForm != null) {
        quickNewContactForm.register(findContactsForm.getListView());
    }
}
```

So when **QuickNewContact** has successfully submitted, *BaseFormPanel.onActionComplete* makes a call to *BaseFormPanel.notifySuccess* which notifies each registered widget by calling *widget.notifySuccess();* from the **FormNotificationInterface**.

findContactsForm.getListView() is a **PartyListView** which implements that interface and its *notifySuccess()* method performs *getStore().reload();*.

Adding a GWT Widget to a Screen

To load a GWT widget, we will need to load the GWT module that contains it in the screen definition's <action> section. There is a context variable called *gwtScripts* which is an array containing paths to each GWT module that should be loaded.

For example, if we wish to load our *accounts* module for all screens in the crmsfa accounts tab, then we can do the following in the **main-section-decorator** of `AccountsScreens.xml`:

```
<screen name="main-section-decorator">
    <section>
        <actions>
            <set field="gwtScripts[]" value="crmsfagwt/org.opentaps.gwt.crmsfa.accounts.accoun
            ....
        </actions>
        <widgets>
            <include-screen name="main-section-template" location="component://opentaps-common
        </widgets>
    </section>
</screen>
```

Here **crmsfagwt** is the mount point of the webapp used to deploy the crmsfa widgets. And **org.opentaps.gwt.crmsfa.accounts.accounts** is the java path for

org/opentaps/gwt/crmsfa/accounts/accounts.gwt.xml (the gwt.xml files contains the module definition and they are included in the JAR on compilation).

Internationalization

Enabling Internationalization

To enable internationalizations, edit the file `hot-deploy/opentaps-common/src/common/org/opentaps/gwt/common/common.gwt.xml` and add the locale that you wish to be included to this list:

```
<extend-property name="locale" values="en"/>
<extend-property name="locale" values="zh"/>
<extend-property name="locale" values="fr"/>
<extend-property name="locale" values="bg"/>
<extend-property name="locale" values="ru"/>
<extend-property name="locale" values="pt"/>
<extend-property name="locale" values="es"/>
<extend-property name="locale" values="nl"/>
<extend-property name="locale" values="it"/>
```

Then, when opentaps is built, the UI labels for that locale will be translated into GWT files (see below.) During development, you can also comment out some of these locales to speed up build time.

Dictionary

A Dictionary is a GWT client-side object which can be constructed from a JavaScript map passed from the server-side. In opentaps, we use the `Dictionary` to pass properties from the server to the client.

First, the properties are defined as a JavaScript array or map on the server side in HTML, such as this example from `hot-deploy/opentaps-common/webapp/common/includes/header.ftl`:

```
<#if gwtScripts?exists>
  <meta name="gwt:property" content="locale=${locale}"/>
  <#-- set up the OpentapsConfig dictionary (see OpentapsConfig.java) -->
  <script type="text/javascript">
    var OpentapsConfig = {
      <#if configProperties.defaultCountryCode?has_content>
        defaultCountryCode: "${configProperties.defaultCountryCode}",
      </#if>
      <#if configProperties.defaultCountryGeoId?has_content>
        defaultCountryGeoId: "${configProperties.defaultCountryGeoId}",
      </#if>
      <#if configProperties.defaultCurrencyUomId?has_content>
        defaultCurrencyUomId: "${configProperties.defaultCurrencyUomId}",
      </#if>
        applicationName: "${opentapsApplicationName}"
    };
  </script>
</#if>
```

Then, an `OpentapsConfig` class is defined with a `Dictionary`:

```
package org.opentaps.gwt.common.client.config;

import com.google.gwt.i18n.client.Dictionary;

public class OpentapsConfig {
    private static final String DEFAULT_COUNTRY_CODE_KEY = "defaultCountryCode";
    // ....
    private static Dictionary dictionary = null;

    public OpentapsConfig() {
        dictionary = Dictionary.getDictionary("OpentapsConfig");
    }

    public String getDefaultCountryCode() {
        return dictionary.get(DEFAULT_COUNTRY_CODE_KEY);
    }
    // ....
}
```

At run time, GWT will automatically map the `OpentapsConfig` JavaScript array to the Dictionary in `OpentapsConfig.java`