Creating custom Apps in LES - 9.1

Introduction

As part of 9.0 services enablement, REFS now supports deploying custom (*usr/var*) apps from LES. This feature allows adding custom app(s) to the existing REFS environment with customized modules, taskflows & tasks. Additionally, support has been added to hotfix these custom apps to deploy them to the multiple REFS environments.

App Folder Structure

Complying with the LES terminology, LESDIR now has usr and var folder structure for custom REFS apps.

```
LESDIR
   - webclient
        - refs
            - usr
                 - customerApp
                     - config
                     - resources
                     - src
                     - build.xml
                     - customer.xml
            - var
                 - servicesApp
                     - config
                     - resources
                     - src
                     - build.xml
                     - services.xml
```

Both **customerApp** and **servicesApp** with sample taskflows are available in LESDIR/samples/web/refs.

These app folders can be copied into the usr and var folder as a starting point for your custom app.

- **config** - handler beans/module/taskflow/task definition XML files
- **resources** - localization files, images and other static resource files
- **src** - javascript source files
- **build.xml** - ant target definition to build the app
- **{app}.xml** - Define csbuilder targets for compilation

Currently, the sample app uses "rp." namespace. If you are planning on building an MVC app, it requires dedicated app namespace. See RPWEB-8720 for details.

App Structure

The app structure can be customized by adding/editing information in the **config** folder and **{appId}.xml.** By default, customer and services sample apps have the following structure:

location	LESDIR/webclient/refs/usr/customerApp	**location**	LESDIR/webclient/refs/var/servicesApp

appId *	customer	**appId ***	services
Module	customer	**Module**	services
Taskflow	RP.Customer.SampleTaskflow	**Taskflow**	RP.Services.VersionTaskflow
Task	sampleTask	**Task**	versionTask
widgetXtype	samplewidget	**widgetXtype**	versionwidget

* should not be be customized

Build Configurations for Custom Apps

Applications should be configured in a hotflix compliant way. The existing web client hotfix process will not work for applications that are not built in a hotfix compliant way. In general, configuring an app to be built correctly is less work than the alternative. The sample xml files found in `%LESDIR%` are hotfix compliant.

Example build config

```
<!-- Sets several key variables -->
<project project-dir="."
         js-dir="${project-dir}/src/js"
         config-dir="${project-dir}/config"
         app-id="${appId}"
         hf-compliant="true">

  <target name="handler-beans"
          type="handler-beans"
          source-dir="${config-dir}/mappings" />
<!-- Process module definition files -->
    <target name="modules"
          type="module"
          copyright="false"
          debug="true"
          source-dir="${config-dir}/modules">
  </target>
<!-- Process taskflow/task informations -->
  <target name="taskflows"
          type="tf_config"
          taskflow-dir="${config-dir}/taskflows"
          task-dir="${config-dir}/tasks"
          >
  </target>
<!-- Process javascript files -->
  <target name="VersionTask"
          type="js"
          source-dir="${js-dir}/Versions">
    <include name="versionTask.js" />
    <include name="versionWidget.js" />
  </target>
<!-- Process database YAML files -->
  <target name="database"
          type="load"
          source-dir="${project-dir}/db">
      <include name="messages" />
      <include name="taskflows" />
  </target>
</project>
```

To make a build configuration hotfix compliant, specify `hf-compliant="true"` as a `project` attribute, specify an `app-id` of *services* or *customer.* Avoid specifying output paths or target dirs unless absolutely necessary. Allow `CSBuilder` to figure out where to put the files on its own using the specified `app-id`. If a path *must* be specified manually (such as for a `copy` target), use relative paths like `deploy/${app-id}/yourCustomPath` and `web/${app-id}/yourCustomPath`.

See this page for more detail on the subject.

Deploying Custom App

If the LESDIR has either **customerApp** or **servicesApp** set up, build LES deploys these apps to your local REFS environment.

Additionally running **ant** from the app folder deploys both the services and customer app into your local REFS environment.

```
> cd %LESDIR%\webclient\refs\var\servicesApp
> ant
```

```
> cd %LESDIR%\webclient\refs\usr\customerApp
> ant
```

For 9.1 & later versions, perform the REFS database upgrade to load the data from the new app.

Hotfixing Custom App

Creating Hotfix

Both service and customer app are web hotfix compliant. The following environment variables are required for creating a services/customer app hotfix:

- DEVTOOLS
- LESDIR
- REFSDIR

Sample command for creating a customer app hotfix:

```
> cd %LESDIR%
> hotfix cpr -product customer -from {FROM_TAG} -to {TO_TAG} -description
{HOTFIX_DESCRIPTION}
```

Sample command for creating a services app hotfix:

```
> cd %LESDIR%
> hotfix cpr -product services -from {FROM_TAG} -to {TO_TAG} -description
{HOTFIX_DESCRIPTION}
```

Please refer to REFS Client Code Hotfixes - <9.3 for more details about creating hotfixes.

Deploying Hotfix

Please refer to the hotfix install instruction file for complete details. The following environment variables are required for deploying a services/customer app hotfix:

- REFSDIR

Sample hotfix deployment:

```
> cd {HOTFIX_FOLDER}
> deploy
```

For 9.1 & later versions, perform database REFS upgrade to load the data from the new app.

Useful Links

- Deploying application code
- App Development
- Module, Taskflow, & Task
- REFS Client Code Hotfixes - <9.3
- The "Hello, World" App