Migrate a DOC Application from 4.0.0 FP2 to 4.0.0 FP3

June 15, 2020

Porting to FP3 from FP2

This document describes how to port a 4.0.0 FP2 application into a 4.0.0 FP3 application.

Main changes:

- JDK8 is not supported anymore. You should now use JDK11. This has an impact on:
  - your development environment. Update your IDE project settings to use JDK11 instead of JDK8.
  - the way you build the docker images that will be used to deploy your application. If you have `Dockerfiles` defined, check that they use the proper JDK. The default `Dockerfile` definition has been updated in the template used to generate applications.
- The Scaffolding Structure has changed
- Scripted Tasks
  - access to the current scenario has been changed. For more information, see engineTask code example in `execution-service-extension/src/.../Tasks.java`
- Web Components:
  - are now delivered as libraries
  - are based on Angular 9.1.x

How To Port/Migrate

If you want to use the new scaffolding organization, it is advised to perform as follows:

1. ensure your FP2 project source is properly committed under GIT
2. generate an empty application using FP3
3. copy the existing code from the FP2 projects to the newly generated ones
4. build the generated FP3 project
5. copy the modifications back from the generated FP3 project to the FP2 and delete the obsolete projects from the FP2 project
6. build the FP2 project
7. commit your changes

Those steps are described in detail in the following sections.

[2] Generate An Empty FP3 Application

Prepare Empty Application Generation

To generate the empty application:

- create a new folder called “fp3”
- copy in that folder
  - the `entities.jdl` file that describes your data model (it can be found in `gene-model/src/main/resources`)
  - the `generator.sh` file provided with FP3 (see section Script for Application Generation of the Installation section of DOC 4.0.0 FP3 Documentation)
  - the `.yo-rc.json` file produced when you originally generated your application.

    Example of `.yo-rc.json` file:

```
{
  "@gene/generator-gene": {
    "promptValues": {
      "projectName": "aircraft-maintenance",
      "projectTitle": "Aircraft Maintenance Optimization",
```

```
      "destination": ".",
      "package": "com.aircraftmaintenance",
      "collectorClass": "AircraftMaintenance",
      "inputFileType": "jdl",
      "inputFile": "./entities.jdl",
    }
  }
}
```

If you do not have `.yo-rc.json` file, create one using the following information:

- `projectName`: this is the `xxx` value that has been used to prefix sub-projects like xxx-libs or xxx-services
- `projectTitle`: this is the title you have defined for your project toolbar
- `package`: this is the java package you have defined
- `collectorClass`: this is the java class name you have defined. You can find it at the beginning of your entities.jdl file

Generate An Empty Application

Use the `generator.sh` command to generate your application.

```
$ cd fp3
```

```
$ ls -al
.yo-rc.json
entities.jdl
generator.sh
```

```
$ ./generator.sh -v 4.0.0-fp3
```

[3] Copy Code from Original FP2 Application to Empty FP3 Application

Refer to the Scaffolding Changes section below, for details on the changes.

Configuration Files

- `build.gradle` update this file to include any specific change you performed
- `settings.gradle`: update this file to include custom projects you may have added (in `xxx-libs` or in `workers`)
- `gradle/templates`: if you edited the contents of this folder, make sure your changes are applied in the new application folder. Note that the `versions.gradle` file has been updated and refers to different version numbers for some of the dependencies.

Deployment

The following services have new configuration items (SPRING_DATA_MONGODB* and SERVICES_PERMISSION_MONGODB*). Make sure to update your `docker-compose` files and `helm` charts if needed :

- `backend-service`
- `data-service`
- `execution-service`
- `scenario-service`

Also note that the data-service now uses web-sockets. The gateway-service now declares the following route (should be presen)t:

```
- id: data_service_websocket
  uri: ws://${services.data.host}:${services.data.port}
  predicates:
      - Path=/websocket/data/**
  filters:
      - StripPrefix=2
```

Source code

- `xxx-libs`: your existing code should be copied as-is to the corresponding projects in the `processing` folder
- `xxx-workers`: your existing code should be copied as-is to the `workers` folder
    - if you created a custom `Dockerfile` for your workers, remember to use JDK11
- `xxx-services/web-frontend-service/src/app/modules`:
    - if you created custom modules,
        - they should be copied into the `web/src/app/modules`

        - imports must be updated in your custom code (see section `Front-end changes` below)
    - if you did not create any customization, you do not need to change anything in the web project.
- `backend-service`: if you created extensions for that service (Routines, ...), copy those files in the `extensions/backend-service-extension` project. You may have to transfer or copy some dependencies in the `build.gradle` script of the target directory.
- `data`: if you created extensions for that service (Custom queries, ...), copy those files in the `extensions/data-service-extension` project.

[4] Build FP3 project

Go to the root of the FP3 project and call the usual commands:

```
$ ./gradlew build
$ ./gradlew docker
```

[5] Merge FP3 changes into FP2 project

During this phase, you will have to delete the now-obsolete FP2 projects

- xxx-libs content
- xxx-services content

Scaffolding Changes

This section describes precisely the project structure changes. The goal of those changes is to further clarify the projects where development may occur:

- `xxx-libs` → `processing` and `extensions`
- `xxx-services/workers` → `workers`
- `xxx-services/web-frontend-service` → `web`

All the other projects should not require any development/modification.

FP2	**FP3**	Comments
xxx-libs • engine • execution-service-extension • model-checker	extensions • **backend-service-extension** • **data-service-extension** • execution-service-extension **processing** • engine • checker	Two new extension points are provided to add • Backend routines • Custom data service extensions model-checker is renamed into checker
xxx-services **workers** • **engine-worker** • **checker-worker**	Contents have been moved to **workers** • **engine-worker** • **checker-worker**	Workers are in a dedicated folder.
xxx-services • **backend-service** gene-services • data • execution-service • gateway-service • scenario-service	gene-services • **backend-service** • data • execution-service • gateway-service • scenario-service	Backend service is included in the main generic services. It is expected that no modification is required within those projects. Extensions should be performed in extensions/*
xxx-services • **web-frontend-service**	**web**	Web frontend coding is separated.

Front-end changes

Imports

`@gene/web-frontend-base` has been split into several libraries. Imports will have to be updated for the following classes (this list may not be complete)

New package	**Imports to update**
@gene/core	AuthenticationService, ApplicationConfiguration, APP_CONFIG
@gene/data	GeneContext, TaskService, NewJobDescription, TaskServiceToken, JobStatus GeneScenarioApiService, PathNode, PathEvent, Property, ScenarioStatus, pathNodesComparator GeneDataAccessService, Access AggregationQueryResult, PaginatedRequest, Page, GeneEntityFilter, GeneEntityFilter, INTERNAL_ID_FIELD
@gene/layout	GeneDashboardWidgetConfiguration
@gene/widget-core	GeneWidgetConfiguration, GeneWidgetManifest, GeneWidgetConfigurationEvent, ValidationUtils, GeneCustomWidgetFactoryService, GeneWidgetConfigurationEventType, GeneWidgetConfigurationEvent
@gene/widget-data	GeneWidgetConfiguration, GeneWidgetManifest, GeneWidgetConfigurationEvent, ValidationUtils, GeneCustomWidgetFactoryService, GeneWidgetConfigurationEventType, GeneWidgetConfigurationEvent
@gene/common-widget	GeneSimpleButtonConfiguration, GeneBaseButtonComponent, GeneDateRendererComponent, GeneNavigationTargetConfiguration, GeneScenarioTimelineComponent
@gene/component	GeneModalService, GenePopoverComponent, GeneLoadingOverlayModule, GeneModalComponent, GeneModalConfiguration, GeneModalDialogButton, CANCEL_BUTTON

Porting custom widgets

Keeping custom style

The custom widgets made for FP2 should work without trouble in FP3. However due to style changes (white background and border on all dashboard widgets), you might want to adjust the style of your widgets. Add the following property to your widgets manifest to avoid using the new style:

```
preventDefaultCss:true
```

This can cause sizing issues and it may be useful to apply the following to the widgets main containers style (if your widget disappears or does not occupy all allocated space on the grid) :

```
  top:0;
   bottom: 0;
   right: 0;
   left:0;
   position: absolute;
```

Using platform style

For a more uniform looking user interface, it is recommended to use the platform's header in place of (if you have any) custom ones.

To do so:

- don't use preventDefaultCss in your manifest
- add the following to your manifest, this will enable configuration for the header properties :

```
hasHeader:true,
hasTitle:true,
hasIcon:true
```

- The style of your widgets main container should probably contain the following :

```
height: 100%;
width: 100%;
```

```
position: relative;
```

- Remove custom header and style code from your widget (and possibly header configuration from your configurator widget)
- Also, remove border styling that will result in double borders using the platform's style
- Make sure that the configuration (hard-coded or returned by the configurator) contains the following. You may have to remove and add again your widget to make sure the configuration is complete :

```
header: {
    visible:true
}
```

GeneCustomWidgetWrapper → GeneDynamicWidget

The gene-custom-widget-wrapper component has been replaced with gene-dynamic-widget. If you used the wrapper, some adjustments will be required

FP2 with wrapper

```
<gene-custom-widget-wrapper>
  <xxx-custom-widget></xxx-custom-widget>
</gene-custom-widget-wrapper>
```

FP3 with a dynamic widget

```
<gene-dynamic-widget
  [manifest]="manifest"
  [widgetConfiguration]="configuration">
</gene-dynamic-widget>
```

Note that you no longer use a custom selector, but pass the manifest and configuration for your custom widget