Simulating Application Logic

You can use JavaScript to simulate application logic that brings your UI to life.

You will need the following files:

- Component file (.qml) that will specify the API of the UI
- JavaScript file that generates mock data for the UI. For more information about using JavaScript, see Integrating QML and JavaScript.
- Module definition file (*qmldir*) that declares the component (QML type) you specify in the UI file. For more information, see Module Definition qmldir Files.

Here, you will create a component based on the QObject class that will be registered as a singleton type. This enables the use of global property values in the UI.

To create the necessary files:

1. In the File Explorer, create a new folder for the mock data inside the *imports* folder in your project folder (for example, *Data*).

   Note: Make sure to capitalize the *Data* folder name because you will need to import it as a component later, and component names must be capitalized.

   Note: If you place this folder somewhere else in the project, you will need to add the path to the list of imports. To do this, in Qt Design Studio, open the project file (.qmlproject) to add the path to the list of plugin directories passed to the QML runtime. For example, if you placed the *Data* folder inside another folder called *backend* in the root of your project, you would add the following:

   ```
   importPaths: [ "imports", "backend" ]
   ```

2. Select **File** > **New File** > **Qt Quick Files** > **Qt Quick File** > **Choose** to add a Qt Quick file that will specify the API of the UI.
3. Follow the instructions of the wizard to create the Qt Quick file in the data folder. In these instructions, the file is called *Values.qml*.

   Note: Make sure to capitalize the filename because it will become a custom component.

data for the UI.

5. Follow the instructions of the wizard to create the JavaScript file in the Data folder. In these instructions, the file is called *simulation.js*.
6. Delete the template text in JavaScript file and save the file.
7. In a text editor such as Notepad, create a module definition file called *qmldir* with the following contents and place it in the data directory:

```
singleton Values 1.0 Values.qml
```

8. Open *Values.qml* in the Code view for editing.
9. Add the following code to the top of the file to register the QObject-derived class that you will use to expose the global properties as a singleton type:

```
pragma Singleton
```

10. Add the following import statement to import the *simulation.js* file to use the functionality that it provides:

```
import "simulation.js" as JS
```

11. Replace the default Item declaration with the following code to create a QObject-derived class that will list the global properties you want to simulate and their default values:

```
QtObject {
    id: values

    // property values to simulate
    property int name1: 5
    property string name2: "foo"
    property real name3: 2.5

}
```

12. Add the following code to use a Timer component to specify a range of values for the property:

```
property Timer name1Timer: Timer{
    running: true
    repeat: true
    onTriggered: JS.name1Timer()
    interval: 10
}
```

This will execute the function defined for `onTriggered` every 10 ms. Within your javascript functions you

Note: You must add the JavaScript method `name1Timer()` to the JavaScript file. You have the option of adding this JavaScript code directly within the `onTriggered` handler as well.

13. Open the .ui.qml file of the Component that will use the simulated data and add the following code to the top of the file in order to import the Data folder as a QML module:

```
import Data 1.0
```

14. Returning to the **2D** view, locate the property that should be bound to the simulated values. Select ⚙ and **Set Binding** for the property and enter the simulated Value property. For example, you would set the following expression to bind to the example `name1` property:

```
Values.name1
```

‹ Loading Placeholder Data Simulating Dynamic Systems ›