Table of Contents

If you want to customize your screen, with flutter widgets on the client, we offer a simple API to change a screen to whatever you need. In the following sections you'll learn how to achieve that.

Creating a custom screen manager

To be able to customize your screens, you have to create a custom screen manager for your application. You can achieve this by extending your dart class from **CustomScreenManager** like so:

```
class ExampleCustomScreenManager extends CustomScreenManager
```

The **CustomScreenManager** has all the functions you'll need to provide your users with customized screens. For this example we'll override both the **init** and the **onMenu** function.

The **onMenu** function is used to provide the menu with screens that aren't provided by the server. To add our own entry to the Menu we just have to override the function like this:

```
@override
void onMenu(SoMenuManager menuManager) {
    menuManager.addItemToMenu(
        id: 'CustomComponentId',
        group: 'Customscreens',
        text: 'Custom Screen',
        image: 'FontAwesome.map',
    );
}
```

This adds an entry to our menu. For us to be able to react to a click on this menu item we have to override the init function:

```
@override
init() {
    SoComponentCreator componentCreator = SoComponentCreator();

    super.registerScreen('CustomComponentId',
CustomScreen(componentCreator));
super.registerScreen('com.sibvisions.example.ContactScreen:L1_MI_DOOPENWORKS
CREEN_COM-SIB-EXA-CONWORSCR', ContactCustomScreen(componentCreator));
}
```

In this case the init function registers custom screen implementations for specific identifiers.

Creating a custom screen

Theres are two use-cases of custom screens

1. A screen which is available only on the client side
2. A screen which is available on the server

In this tutorial we'll cover both use-cases. Let's start with a custom screen thats not available on the

server. For this we have to extend an already existing class: **CustomScreen**.

```
class CustomScreen extends CustomScreen {
    CustomScreen(SoComponentCreator componentCreator) :
super(componentCreator);

    @override
    Widget getWidget() {
    return CustomWidget();
    }

    @override
    void update(Request request, ResponseData responseData) {}

    @override
    bool withServer() {
        return false;
    }
}
```

In this class you can return any widget you want in the **getWidget** function. The **withServer** function tells the client, not to communicate with the server for this screen. The **update** function is not necessary for this example.

If you want to create a custom screen that communicates with the server, you have to `return true;` in the **withServer** function:

```
class ContactCustomScreen extends CustomScreen {
    ContactCustomScreen(SoComponentCreator componentCreator) :
super(componentCreator);

    @override
    Widget getWidget() {
        CoCustomComponent contactComp = new CoCustomComponent(
    GlobalKey(debugLabel: 'contact'),
    componentScreen.context,
    );

    contactComp.widget = Text('This is my replaced widget');

    CoPanel comp =
this.componentScreen.getComponentFromName('contactPanel');
    this.componentScreen.replaceComponent(comp, contactComp);

    IComponent component = this.componentScreen.getRootComponent();

    if (component != null) {
    return component.getWidget();
    } else {
        return Container(
            alignment: Alignment.center,
```

```
        child: Text('No root component defined'),
            );
        }
    }

    @override
    void update(Request request, ResponseData responeData) {
        componentScreen.updateData(request, responeData);
    if (responeData.screenGeneric != null) {
componentScreen.updateComponents(responeData.screenGeneric.changedComponents
);
        }
    }

    @override
    bool withServer() {
        return true;
    }
}
```

In the **getWidget** function we return our custom widget. In this case we create a CoCustomComponent which can hold any widget we want. Afterwards we get the component that we want to replace from the widget tree via `this.componentScreen.getComponentFromName('contactPanel');`.

To replace this component with our own widget, we just have to call `this.componentScreen.replaceComponent(comp, contactComp);`.

We also override the **update** function to update our components when we get new data from the server. The **withServer** function now tells our screen to communicate with the server.

An example screen can be found here and an example custom screen manager here.