Custom App Development for the Coatmaster Flex

This guide provides everything you need to know to create your own QML applications for the **Coatmaster Flex** coating thickness measurement device. These custom apps, which appear as dialogs on the device, allow you to build tailored interfaces for controlling processes like your coating line, enabling a seamless workflow for measuring thickness and adjusting parameters directly on the device.

Deployment

To deploy your custom app, you must package all your files into a single `.zip` archive.

- **Main File:** The main QML file of your application **must be named** `App.qml`.
- **File Structure:** The `App.qml` file must be located at the **root level** of the zip archive, not inside any folders. Other assets like additional QML components or images can be placed in sub-folders (e.g., `/lib`, `/images`).
- **Uploading:** The final `.zip` file is then uploaded to the Coatmaster Flex through its web interface.

I. Core Concepts & API Reference

This section covers the foundational components and their properties.

Navigation: The Grid System

The user interface is navigated using a simple grid system. Every interactive `FlexQmlItem` must be assigned a `navigationRow` and a `navigationColumn` to place it within this grid. The user can then move between items using the directional keys on the device.

For example, in `ExampleApp.qml`, you can see a `FlexTextInput` and a `FlexCheckbox` placed on the same row but in different columns.

```
FlexTextInput{
    navigationRow: 1
    navigationColumn: 0
    // ...
}

FlexCheckbox{
    navigationRow: 1
    navigationColumn: 1
    // ...
}
```

The `FlexDialog` Object

The `FlexDialog` is a global object that provides access to system-level properties and functions within your QML app.

Properties:

- `unit` (string, read-only): The measurement unit configured on the device (e.g., "µm").
- `foregroundColor` (color, read-only): The primary foreground color for text and icons.
- `backgroundColor` (color, read-only): The dialog's background color.
- `accentColor` (color, read-only): The accent color used for highlighting focused items.

Invokable Functions:

- `closeDialog()`: Closes the current dialog.
- `loadQmlDialog(const QUrl& url, QWidget* parentDialog=nullptr, QVariantMap context=QVariantMap())`: Loads and opens a new QML dialog.
  - `url`: The URL to the QML file for the new dialog.
  - `parentDialog`: (Optional) A reference to the parent dialog.
  - `context`: (Optional) A `QVariantMap` to pass properties to the new dialog's context.

The `FlexQmlItem` Component

Since the Coatmaster Flex does not use a mouse or touchscreen, every interactive element in your app must be a `FlexQmlItem`. This is the foundational component that enables user interaction. Our pre-built components like `FlexButton` already incorporate a `FlexQmlItem`.

Properties:

- `navigationRow` (int): The row position of the item in the navigation grid.
- `navigationColumn` (int): The column position of the item in the navigation grid.
- `isFocused` (bool): `true` if the item currently has navigation focus. A highlight frame is automatically shown.
- `visualItem` (QQuickItem*): A reference to the underlying visual QML item.
- `itemId` (string): A unique identifier for the item.
- `foregroundColor`, `backgroundColor`, `accentColor` (color, read-only): Inherited from `FlexDialog` for consistent theming.

- `selectable` (bool): If `true`, the item can receive focus.
- `keyCapture` (bool): If `true`, this item will capture all key events (up, down, left, right, ok, back) until set to `false`. This is useful for components like lists or combo boxes that need to handle navigation internally.
- `visible` (bool): Controls the visibility of the item.
- `enabled` (bool): If `false`, the item is visible but cannot be focused or activated.
- `text` (string): A text property often used for passing data to the underlying visual item or for opening the keyboard with initial text.
- `flexDialog` (QObject*): A reference to the `FlexDialog` object.

Invokable Functions:

- `openKeyboard(string title="", string type="normal")`: Opens the on-screen keyboard.
  - `title`: An optional title to display on the keyboard dialog.
  - `type`: The keyboard type. Can be `"normal"`, `"numeric"`, or `"full"`.

II. Pre-Built UI Components

To make development faster and easier, we provide a set of ready-to-use QML components that are built upon `FlexQmlItem`.

`FlexTextInput`

A text input field. When activated, it calls `openKeyboard()`.
properties:

- `navigationRow` (int): The row position of the item in the navigation grid.
- `navigationColumn` (int): The column position of the item in the navigation grid.
- `keyboardType` (string -> "normal", "numeric" or "full")
- all properties of QtQuick.Controls TextField

`FlexButton`

A standard button. Its `onClicked` signal is triggered on activation.
properties:

- `navigationRow` (int): The row position of the item in the navigation grid.
- `navigationColumn` (int): The column position of the item in the navigation grid.
- all properties of QtQuick.Controls Button

`FlexComboBox`

A dropdown menu. It uses `keyCapture` to handle internal navigation of its options.
properties:

- `navigationRow` (int): The row position of the item in the navigation grid.
- `navigationColumn` (int): The column position of the item in the navigation grid.
- all properties of QtQuick.Controls ComboBox

`FlexList`

A list of items. It uses `keyCapture` to allow the user to scroll through its contents.
properties:

- `navigationRow` (int): The row position of the item in the navigation grid.
- `navigationColumn` (int): The column position of the item in the navigation grid.
- `keyCapture` (bool): If `true`, this item will capture all key events (up, down, left, right, ok, back) until set to `false`. This is useful for components like lists or combo boxes that need to handle navigation internally.
- all properties of QtQuick.Controls ListView

`FlexCheckbox`

A simple checkbox that toggles its `checked` state on activation.
properties:

- `navigationRow` (int): The row position of the item in the navigation grid.
- `navigationColumn` (int): The column position of the item in the navigation grid.
- all properties of QtQuick.Controls CheckBox

III. Measurements and Device Interaction

Your app can directly interact with the device's measurement capabilities and hardware buttons.

Taking a Measurement with `FlexQmlMeasure`

The `FlexQmlMeasure` component is your primary tool for handling measurements.

1. **Initiate a Measurement**: Call the `measure()` function on your `FlexQmlMeasure` item.
2. **Displaying Results**: The `onNewMeasurement` signal is emitted when a measurement is complete. It returns a `measurement` object with the data.
3. **Handling Busy State**: The `busy` property of `FlexQmlMeasure` is `true` while a measurement is in progress.

The `measurement` Object

The `onNewMeasurement` signal provides a `measurement` object with the following structure:

- `configId` (int): The ID of the measurement configuration used.
- `displayStatus` (object): An object containing display information.
  - `category` (string): The status category (e.g., "OK").
  - `colourCode` (string): A hex color code for displaying the result.
  - `icon` (string): The name of an icon related to the status.
  - `showThickness` (bool): Whether the thickness value should be displayed.
  - `text` (string): Additional status text to display.
- `error` (int): An error code (0 if no error).
- `evaluateState` (string): The evaluation state (e.g., "OK").
- `fit` (double): The fit value of the measurement.
- `fitState` (string): The state of the fit calculation.
- `id` (int): The unique ID of the measurement record.
- `sampleId` (int): The ID of the sample.
- `snr` (double): The signal-to-noise ratio.
- `snrState` (string): The state of the signal-to-noise ratio.
- `thickness` (double): The raw thickness value.
- `thicknessString` (string): The thickness value formatted as a string.
- `timeStamp` (string): The timestamp of the measurement.

Responding to Hardware Buttons

The `FlexDialog` object allows you to listen for hardware button presses.

- **`onTriggerPressed`** : Fired when the main measurement trigger is pressed.
- **`onKeyBackPressed`** : Fired when the back key is pressed. A common use is to close the custom app dialog by calling `FlexDialog.closeDialog()` .
- **Emergency Exit**: If a QML app is unresponsive, a **long press of the back button** will force-close the dialog and return to the main menu. This is a safety feature to prevent getting stuck in a broken app.

IV. Fetching Data from a Server

Your app can retrieve data from the Coatmaster server using HTTP requests.

Using the `utils.js` Library

We provide a utility library, `utils.js` , for making asynchronous network requests. You must import it in your QML file: `import "../lib/utils.js" as Utils` .

Security Note: For security reasons, QML apps can only make HTTP calls to the Coatmaster server they are running on. The base URL is selected automatically. You only need to provide the path to the endpoint (e.g., `/configurations`).

The library provides the following functions:

- `Utils.httpRequest(method, path, requestData)`
- Convenience Functions: `Utils.get(path)` , `Utils.post(path, data)` , `Utils.put(path, data)` , `Utils.patch(path, data)` , `Utils.del(path)` .

Accessing a Remote Server

It is also possible to access any remote server through a built-in proxy. To do this, you must construct the URL in the following format:

`"http://localhost:9883/proxy?target=<your_full_remote_url>"`

For example, to access `https://jsonplaceholder.typicode.com/posts/1` , you would use the URL:

`"http://localhost:9883/proxy?target=https://jsonplaceholder.typicode.com/posts/1"`

You can use the standard `XMLHttpRequest` object for this purpose.

Remote Server Example:

```
Component.onCompleted: {
    var xhr = new XMLHttpRequest();
    var url = "http://localhost:9883/proxy?target=[https://jsonplaceholder.typicode.com/posts/1]
```

```
(https://jsonplaceholder.typicode.com/posts/1)";

    xhr.open("GET", url);

    xhr.onreadystatechange = function() {
        if (xhr.readyState === XMLHttpRequest.DONE) {
            try {
                var jsonData = JSON.parse(xhr.responseText);
                responseText.text = JSON.stringify(jsonData, null, 2);
            } catch (e) {
                responseText.text = xhr.responseText;
            }
        }
    }

    xhr.send();

}

Text {
    id: responseText
    text: "Loading..."
}
```

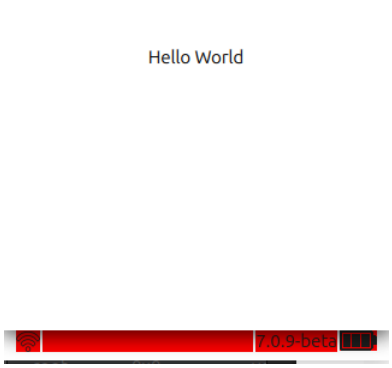
V. Practical Walkthroughs

Minimal Working Example (`MinimalApp.qml`)

This is the simplest possible app. It displays "Hello World" and ensures the dialog can be closed with the back button.

```
// Import necessary QML modules and custom components.
import QtQuick 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15
import "../lib/utils.js" as Utils
import "../lib"
import FlexUi 1.0

// The root item of the application.
Item {
    id: app
    anchors.fill: parent
    property string appId: ""

    Text {
        text: "Hello World"
        anchors.centerIn: parent
        color: FlexDialog.foregroundColor // Use system color
    }

    // --- Connections: Handle global device events ---
    Connections {
        target: FlexDialog

        // This is essential to allow the user to close the dialog.
        onKeyBackPressed: {
            FlexDialog.closeDialog();
        }
    }
}
```

Screenshot



Full Example (`ExampleApp.qml`)

This fully commented example demonstrates how to structure an app, fetch data, handle user input, and perform measurements on the Coatmaster Flex.

```
// Import necessary QML modules for UI, layouts, and controls.
import QtQuick 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15
// Import the JavaScript utility library for making HTTP requests.
import "../lib/utils.js" as Utils
import "../lib"
// Import the custom Flex UI components.
import FlexUi 1.0


// The root item of the application.
Item {
    id: app
    // A property to track if the app is busy loading data.
    property bool isLoading: false
    anchors.fill: parent
    property string appId: ""

    // --- FlexQmlMeasure: Handles all measurement logic ---
    FlexQmlMeasure {
        id: measureItem

        // This signal is triggered when a new measurement is completed.
        onNewMeasurement: function (measurement) {
            if (measurement && 'displayStatus' in measurement) {
                var displayStatus = measurement.displayStatus;
                // Set the color of the result text based on the status.
                if ('colourCode' in displayStatus && displayStatus.colourCode) {
                    resultThicknessText.color = displayStatus.colourCode.trim();
                } else {
                    resultThicknessText.color = FlexDialog.foregroundColor;
                }
                // Display the thickness if available.
                if ('showThickness' in displayStatus && displayStatus.showThickness === true) {
                    resultThicknessText.text = measurement.thicknessString + " " + FlexDialog.unit;
                    messageItem.text = "";
                } else {
                    resultThicknessText.text = "-.-";
                }
                // Display any additional status text.
                if ('text' in displayStatus && displayStatus.text) {
                    messageItem.text = displayStatus.text;
                }
            } else {
                resultThicknessText.text = "-.-";
            }
        }

        // This signal is triggered if an error occurs during measurement.
        onError: function(message) {
            messageItem.text = message;
        }
```

```
}

// --- Connections: Handle global device events ---
Connections {
    target: FlexDialog
    // Trigger a measurement when the hardware trigger is pressed.
    onTriggerPressed: {
        measureItem.measure();
    }
    // Close the app when the hardware back button is pressed.
    onKeyBackPressed: {
        FlexDialog.closeDialog();
    }
}

Image {
    anchors.fill: parent
    source: "coatmasterFlex.png"
    fillMode: Image.PreserveAspectCrop
    asynchronous: true
    opacity: 0.3
}

// A busy indicator that is visible when the app is loading or measuring.
BusyIndicator {
    anchors.centerIn: parent
    running: app.isLoading || measureItem.busy
    visible: running
    z: 2 // Ensure it's on top of other elements.
}

// --- Component Initialization ---
Component.onCompleted: {
    // Set loading to true while we fetch initial data.
    app.isLoading = true;
    // Fetch the list of configurations from the server.
    Utils.httpRequest("GET", "/configurations").then(function (data) {
        // Map the server response to the format needed by the ListModel.
        var modelData = data.map(q => ({
                                           "id": q.id,
                                           "name": q.name
                                       }));
        // Append each item to the ListModel for the ComboBox.
        modelData.forEach(q => appListModel.append(q));
        // Set loading to false now that data is loaded.
        app.isLoading = false;
        // Set the initial selection in the ComboBox.
        appListView.currentIndex = 0
    });

    // Populate a local model for the FlexList.
    lineModel.append({"value": 1, "name": "line A"});
    lineModel.append({"value": 2, "name": "line B"});
    lineModel.append({"value": 3, "name": "line C"});
}

// --- Data Models ---
ListModel { id: appListModel }
ListModel { id: lineModel }

// --- UI Layout ---
GridLayout {
    columns: 2
    anchors.fill: parent

    Text {
        text: "demo app"
        Layout.row: 0; Layout.column: 0; Layout.columnSpan: 2
        font.pointSize: 24
        horizontalAlignment: Qt.AlignHCenter
        Layout.fillWidth: true
    }

    // Text to display the measurement result.
    Text {
        id: resultThicknessText
        text: ""
        Layout.row: 1; Layout.column: 0; Layout.columnSpan: 2
        font.pointSize: 44; font.bold: true
        horizontalAlignment: Qt.AlignHCenter
        Layout.fillWidth: true
    }
```

```
        // A list displaying the 'lineModel'.
        FlexList {
            id: appListView
            navigationRow: 0; navigationColumn: 0
            Layout.row: 2; Layout.column: 0; Layout.columnSpan: 2
            Layout.fillWidth: true; Layout.fillHeight: true
            model: lineModel

            delegate: ItemDelegate {
                width: parent.width
                text: name
                highlighted: appListView.currentIndex === index
            }
            onSelected: function (index, app) {
                // This is where you would handle a selection event from the list.
            }
        }

        // An input field for text.
        FlexTextInput {
            navigationRow: 1; navigationColumn: 0
            Layout.row: 3; Layout.column: 0
            Layout.fillWidth: true
            text: "standard input"
        }

        // A checkbox.
        FlexCheckbox {
            id: checkBox
            navigationRow: 1; navigationColumn: 1
            Layout.row: 3; Layout.column: 1
        }

        // A dropdown to select a measurement configuration.
        FlexComboBox {
            id: myCombobox
            navigationRow: 2; navigationColumn: 0
            Layout.row: 4; Layout.column: 0; Layout.columnSpan: 2
            Layout.fillWidth: true
            currentIndex: 0
            model: appListModel
            textRole: "name"
            delegate: ItemDelegate {
                width: parent.width
                text: name
                highlighted: myCombobox.currentIndex === index
            }
            // When an item is selected, update the appId on the measureItem.
            onSelected: function (index, app) {
                measureItem.appId = app.id
            }
        }

        // An input field for numbers.
        FlexTextInput {
            navigationRow: 3; navigationColumn: 0
            Layout.row: 5; Layout.column: 0
            Layout.fillWidth: true
            keyboardType: "numeric"
            text: "345.56"
        }

        // A button to trigger a measurement manually.
        FlexButton {
            navigationRow: 3; navigationColumn: 1
            Layout.row: 5; Layout.column: 1
            text: "measure"
            // The button is only enabled if a configuration has been selected.
            enabled: measureItem.appId >= 0
            onClicked: {
                measureItem.measure();
            }
        }

        // Display the currently selected appId for debugging/info.
        Text { text: "current appId:"; Layout.row: 6; Layout.column: 0 }
        Text { text: measureItem.appId; Layout.row: 6; Layout.column: 1 }

        // A text area for displaying messages or errors.
        Text {
            id: messageItem
            text: ""
            Layout.row: 7; Layout.column: 0; Layout.columnSpan: 2
```

```
            wrapMode: Text.Wrap
            Layout.fillWidth: true
        }
    }
}
```

Screenshot