Advanced Mobile App Troubleshooting Using Chrome

This topic describes an advanced mobile app troubleshooting scenario where the debugging features of OutSystems cannot be used (e.g. in a Production environment). Check Debugging Applications for more information on using OutSystems debugging features to debug your web and mobile apps.

You can troubleshoot issues in your mobile app using Google Chrome’s Developer Tools which provides you with a set of features to analyze what’s happening under the hood of your application and to debug the JavaScript code generated by the platform. This can be done using a preview of your mobile app on the desktop browser (if possible) or by connecting the desktop browser first to a mobile device running your mobile app.

There are a few tips that can streamline your troubleshooting process while using Chrome Developer Tools. Check the Troubleshooting Tips section below for more information.

Troubleshoot Using Chrome

You can troubleshoot mobile app issues in your Chrome desktop browser if you're not using native plugins or if all native plugins have fallbacks in place that allow your mobile app to work properly outside a real mobile device.
What Can I Troubleshoot?

Since the client code is running on the browser, you can use the development tools of the browser to debug it. You can debug:

- Client Screen Actions
- Client Actions
- Event Handlers
- On Exception Handlers

Check the Generated JavaScript Code Architecture section to learn more about the JavaScript code generated by the platform.

Start Troubleshooting

You can troubleshoot your mobile app as soon as you publish it, checking its logic, screens and viewing the local storage. To do it:

1. Open the Chrome browser on the desktop (use the latest version).
2. Open Chrome Developer Tools (e.g. by pressing F12) and start debugging or troubleshooting.

Check the Developer Tools JavaScript Debugging Reference from Google for more information on the debugging features provided by Chrome Developer Tools.

To debug a client action you can hold down the CTRL key while using the application to trigger that action; this will automatically breakpoint in any client action that is executed. Check below for other useful tips while using Chrome Developer Tools.

Viewing Local Storage Data

You can view data in the local storage while troubleshooting your app in the desktop Chrome browser.

To view the data of a local entity:

1. While previewing your mobile app in the (desktop) Chrome browser, open Chrome Developer Tools (e.g. by pressing F12).
2. Go to the Application tab and navigate to your app’s module WebSQL/<YourModuleName>.
3. Expand this tree node to show a list local entities defined in the application.
4. Select one to view its data.

The tables for the local entities of the application are prefixed by `OSUSR_`, while OutSystems tables are prefixed by `OSSYS_`

The `OSSYS_ENTITY` table stores internal information about each one of the application tables and the `OSSYS_LOG` is used to store logs locally when no network connection is available.

### Running queries

To run queries over local entities:

1. Select `<YourModuleName>` in the tree on the left.
2. Write your SQL queries in the console available on the right.
Troubleshooting on Your Device Connected to the Desktop Browser

You can troubleshoot your mobile app while running it on a mobile device, either through OutSystems Now or by installing it previously through an application package. This is useful to troubleshoot plugin behavior in your mobile app.

This kind of troubleshooting on mobile devices is only possible for Android apps.

Connecting the Desktop Browser to Your Device

Pre-requisites:

- Your mobile device has Android version 4.0 or higher.
- You have Google Chrome version 32 or higher installed in the PC.
- The desktop Chrome version is higher than the Chrome version in the mobile device.
- If the mobile app was installed in the mobile device through an application package, it must have been generated with a Debug build type.

To enable debugging on your device:

1. Open Settings and find the option Developer Options.

   Note: If you don’t see this option listed, take the following steps:
   1. In the device, go to Settings and select About.
   2. Tap seven times on top of the build number.

2. After that, the device will display the Developer Settings option. If this doesn't work, check how to do it in the documentation.

3. Select the Settings > Developer Options.

4. Slide the enable slider to On.

5. Enable the USB Debugging option.

6. Accept the security popups.
To connect your device to the desktop Chrome browser:

1. Connect the device to your PC through USB.

2. If there an option in the device to set the USB mode, make sure it’s set to PTP (or Camera mode).

3. Open the desktop Chrome browser and navigate to the following URL:
   chrome://inspect/#devices

   *Note:* If there are no devices listed, you are probably missing the USB drivers on your computer.

4. Click on the inspect link next to the URL of the WebView.

You can now inspect and debug the contents of the WebView displaying your app in the device.

Check the Developer Tools JavaScript Debugging Reference from Google for more information on the debugging features provided by Chrome Developer Tools.

If your mobile app uses only supported plugins that are included with OutSystems Now, you can connect the desktop browser to your device by following the steps outlined above and troubleshoot it through the OutSystems Now mobile app. This way you will avoid having to install your mobile app through an application package.
Troubleshooting Tips

You can find below a series of tips that may improve your troubleshooting steps:

• To debug a client action, hold down the CTRL key while using the application to call that action. This will automatically breakpoint in any client action that is executed.

• To make the JavaScript code more perceivable and simple to read, click on the Pretty Print button available in the Sources tab.

• To obtain more readable stack traces for asynchronous actions check the Async checkbox available in the Sources tab.

• Keep a look for the errors in the Console tab. You can click on the link on the right side to navigate to the JavaScript location where the error is happening.

• When the application redirects you to an error page the logs in the Console tab disappear. To preserve them, check the Preserve log option.

• Use the Performance tab in Chrome’s Developer Tools for doing performance troubleshooting. If you detect long wait times without any CPU or network usage, this means that your app is probably waiting for data (aggregate or data action results) or waiting for a native plugin operation to complete.
To simulate poor network conditions you can enable the throttling feature available in the Network tab. Be aware that you might get unexpected results when using throttling in Chrome and testing with a real device connected to the browser. Alternatively, you can test your mobile app on a device with its mobile network settings configured for 2G mode only.

Generated JavaScript Code Architecture

A mobile app generates code and assets that run on the server and on the client (mobile device). At the client side, this code is JavaScript and has a specific architecture. This document explains what are the files, their roles, and the generated code for each flow element defined for an action in the Service Studio. This content is useful to know when you’re troubleshooting or checking the generated code at the client side.

Generated JavaScript Files

To locate the JavaScript files generated for client, expand the scripts folder in Chrome DevTools:

The explanation about the highlighted sections is in the table below. Please note that the number of generated JavaScript files may change and that the information provided may not be up to date.

[Image of a tree structure showing folders and files under 'scripts']
## JavaScript Files Roles

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module files</strong></td>
<td></td>
</tr>
<tr>
<td><code>&lt;ModuleName&gt;.controller.js</code></td>
<td>Global client actions of the <code>&lt;ModuleName&gt;</code> module.</td>
</tr>
<tr>
<td><code>&lt;ModuleName&gt;.entities.js</code></td>
<td>SQL code to manipulate the local entities of the <code>&lt;ModuleName&gt;</code> module.</td>
</tr>
<tr>
<td><code>&lt;ModuleName&gt;.model.js</code></td>
<td>Definitions for the <code>&lt;ModuleName&gt;</code> module’s local entities (including referenced entities).</td>
</tr>
<tr>
<td><code>&lt;ModuleName&gt;.languageResources.js</code></td>
<td>Validation messages defined in the <code>&lt;ModuleName&gt;</code> module.</td>
</tr>
<tr>
<td><strong>UI Flow files</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

https://success.outsystems.com/Documentation/11/Developing_an_Application/Troubleshooting_Applications/Advanced_Mob...
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ModuleName&gt;.&lt;UIFlowName&gt;.controller.js</td>
<td>Exception handler of the &lt;UIFlowName&gt; flow</td>
</tr>
<tr>
<td>Screen file, Block file</td>
<td>Contains the actions and events, definition of variables and widget rendering code for the screen/block element.</td>
</tr>
<tr>
<td>&lt;ModuleName&gt;.&lt;UIFlowName&gt;.&lt;ScreenName&gt;.mvc.js</td>
<td></td>
</tr>
<tr>
<td>&lt;ModuleName&gt;.&lt;UIFlowName&gt;.&lt;BlockName&gt;.mvc.js</td>
<td></td>
</tr>
</tbody>
</table>

How Are Flow Nodes Represented in JavaScript

Aggregate

```javascript
// Aggregate: GetLocalProperties

var getLocalProperties = function() {
  var executeQuery = function() {
    var targetlistType = CwSourceReportingP1Model.LocalPropertiesRecordList;
    var result = {
      list: null,
      count: OutSystems_INTERNAL_DataTypes_LongInteger.defaultValue
    }; 
    var countArgList = new OutSystems_INTERNAL_Entities_QueryArgumentList();
    if ((this.records) []
    maxRecords = 0;
    };
    var finaliseSelect = false;
    var selected = "";
    var fromWhereAreWe = "";
    var groupNotSupportedException = "";
    var orderBySql = "";
    var argList = new OutSystems_INTERNAL_Entities_QueryArgumentList();
    var selectColumn = "SELECT ";
    if (containsSelect) 
      selectColumn = "UPDATE ";
    var selectSql = "UPDATE LocalProperties ("ID" 0, ELocalProperties ("LocalProperties.");
    fromWhereAreSql = " FROM "MAIN".", " LOCAL_LOCAL_PROPERTIES (" ELocalProperties.");
    if (maxRecords > 0) {
      orderBySql = " LIMIT ";
      orderBySql = maxRecords;
    };
    var sql = selectSql + fromWhereAreSql + groupNotSupportedException + orderBySql;
    return OutSystems_INTERNAL_Entities_webSQLInstance.executeQuery(sql, CwSourceReportingP1Model.LocalPropertiesRecordList, function(result) {
      var IndexRef = {
        value: 0
      };
    })
    if (strOrderBy = ["localPropertiesAdder", ]); return result
    if (strOrderBy = ["localPropertiesAdder", ]
    result = resultList;
    result = resultList;
    result = resultList;
    result = resultList;
    return controller.executeQuery(function() {
      return glocalProperties.
      (result)
}));
```

https://success.outsystems.com/Documentation/11/Developing_an_Application/Troubleshooting_Applications/Advanced_Mob...
### Refresh Data

```javascript
return flow.executeAsyncFlow(function() {
    // Refresh Query: GetLocalCases
    var result = controller.dataFetchActions.getLocalCasesAggrRefresh(50);
    model.flush();
    return result;
});
```

### Run Client Action

```javascript
return CarAccidentReportingP10Controller.default.clientActions.CreateNewLocalCaseAction[0].value;
```
Run Server Action

```javascript
// Execute Action: UpdateServerCase
model.flush();
return controller.serverActions.updateServerCaseAction(model.variables.getPolicyByInAppgr().listOut.getsetCurrent(callCom
```

JavaScript

```javascript
function callSyncOnLoginJS(pParameters, pSections) {
    if (window.appSync && window.appSync.onLogin) {
        pParameters.WasSyncTriggered = true;
        window.appSync.onLogin();
    } else {
        pParameters.WasSyncTriggered = false;
    }
    return pParameters;
}

callSyncOnLoginJSresult = controller.safeExecuteJSNode(callSyncOnLoginJS, "callSyncOnLogin", "loginOnClick", {
    WasSyncTriggered: false
});
```
Assign

// GPSLocation.Id = CreateNewCase.GPSLocationId
model.variables.GPSLocationVar.idAttr = createNewCaseVar.GPSLocationIdOut.value;

// Case.Id = CreateNewCase.CaseId
model.variables.caseVar.idAttr = createNewCaseVar.caseIdOut.value;

// Case.GPSLocation = CreateNewCase.GPSLocationId
model.variables.caseVar.gpsLocationAttr = createNewCaseVar.gpsLocationIdOut.value;

If

// WasSyncTriggered?
if (callSyncOnLogin.WasSyncTriggered) {

// IsExecuting = False

Switch

```csharp
if ((caseTypeLocal.value.idAttr == CarAccidentReportingP10Model.staticEntities.caseType.FlatTyre)) {
    extraCostOut.value = 50;
} else {
    if ((caseTypeLocal.value.idAttr == CarAccidentReportingP10Model.staticEntities.caseType.fuel)) {
        extraCostOut.value = 75;
    }
}
```
For Each

```
// Foreach Photos
    try {
        var photosIterator = callContext.iterationContext.getIterator(model.variables.photosVar);
        var photosIndex = 0;
        while (photosIndex < model.variables.photosVar.length) {
            photosIterator.currentRowNumber = photosIndex;
            if (model.variables.photosVar.get(model.variables.photosVar).positionIdAttr == photoPc)
                OutSystems.Internal.Builtin.SystemActions.listRemove(model.variables.photosVar, model.variables.photo
            photosIndex++;
        }
    } finally {
        callContext.iterationContext.registerIterationEnd(model.variables.photosVar);
    }
```

Destination

```
// Destination: /CarAccidentReportingP10/HomePage
```
Raise Exception

```javascript
throw new OutSystems.Internal.Exceptions.UserException("CarAccidentReportingUI01.InvalidLogin", "Invalid Username or Password");
```

Exception Handler

```javascript
try {
  // Handle Error: AllExceptions
  if (!OutSystems.Internal.Exceptions.isSystem(ex)) {
    OutSystems.Internal.Logger.error(null, ex);
    return flow.executeAsyncFlow(function() {
      // isExecuting = false
      model.variables.isExecuting value = false;
      return flow retornaSync();
    });
  }
} catch (ex) {
  throw ex;
}
```