# MYOB Advanced

## Customising the Mobile App - Advanced

Last Updated: 16 October 2019

# Contents

# Introduction

This document introduces extended functionality to customise the MYOB Advanced mobile app by using MYOB Advanced Customisation Platform.

The document is intended for developers who are learning how to customise the MYOB Advanced mobile app or other MYOB Advanced Framework-based mobile apps. It is based on an end-to-end customization task that demonstrates the general approach to customizing the MYOB Advanced mobile app. The document consists of three parts, which take a total of about six hours to complete.

After you complete the document, you will have an understanding of how to perform advanced customization tasks of the MYOB Advanced mobile app or other apps developed with MYOB Advanced Framework. Upon completion of the document, you will have learned how to perform the following types of customization activity:

- Mapping a new screen from the beginning
- Organizing the UI elements on a screen
- Configuring redirection from one screen to another
- Using form customization to customize the mobile app

# Preparing the Environment

In this training, you will learn how to customize the MYOB Advanced mobile app. Before you start the practical steps of the document, you have to prepare the development environment.

## Required Role for Users

Specialists who will work on customization projects should be assigned the Customizer role in the application instance that is to be customized and tested, as well as in the production application that should be updated with the customization project. With this role assigned, the specialists can use the customization tools and facilities of the MYOB Advanced Customization Platform, and upload and publish customization projects.

A user with the Administrator role can assign the Customizer role to the needed users by using the Users (SM201010) form.

## Preparation of a Mobile Device

To test a customized mobile app, you will have to install an MYOB Advanced mobile app to a mobile device that is connected to the same local network as the computer hosting the development environment of your MYOB Advanced instance.

# Part 1: Configuring a Screen

In this part, you will add a screen that corresponds to the Invoices (SO303000) form and map it from the beginning. This includes the following actions:

- Mapping the screen's fields and actions
- Organizing the fields of the screen into a layout
- Configure headers and groups for the fields
- Configure tabs and data tabs

## Preparing the Mobile Site Map

Before you start configuring the Invoices (SO303000) screen for the mobile app, prepare the customization project and add the Invoices screen to the mobile site map as follows:

1. In your MYOB Advanced instance, create a new customization project, and open it in the Customization Project Editor.
2. On the Mobile Application page of the Customization Project Editor, add the Invoices screen to the mobile site map.
3. On the Mobile Application page of the Customization Project Editor, update the main menu of the mobile app so that it contains a tile for the Invoices screen.

## Lesson 1.1: Adding Fields and Actions

In this lesson, you will start configuring the Invoices (SO303000) screen by adding to it fields and several actions that can be performed on this screen. After reviewing information on the types of actions, you will add some fields and actions. Finally, you will test the implemented screen on a mobile device.

### Types of Actions

An action can be performed under the following types of objects or groups of MSDL objects:

- container
- list
- record
- selection

Depending on which object the action is applied to, MSDL includes the following MSDL objects:

- containerAction: Applied to the whole container
- listAction: Applied to the whole list of records
- recordAction: Applied to the opened record
- selectionAction: Applied to the items selected in a list

You can learn what actions are defined for a form by exploring the WSDL schema of the form. But the WSDL schema does not indicate which action is applied to a container, which is applied to a list, which is applied to the opened record, and which is applied to selected items. You should understand what the action is applied to by learning what the

desired action on a form does. For example, the Save and Cancel actions are applied to the record as a whole, so you should map them as recordAction objects. On the other hand, the Insert action adds a new record in the container, so you should map the Insert action as a containerAction object. Note that container actions are defined inside the corresponding containers.

**Note:** You must declare the Cancel action for all screens that include it in the WSDL schema. Without the Cancel action mapped, the changes discarded in the mobile app might not be discarded on the server.

## Step 1.1.1: Add Fields to the Invoices Screen

To add fields to the Invoices (SO301000) screen, do the following:

1. Open the WSDL schema of the Invoices form. Find and explore the following complex types:
   - InvoiceSummary
   - DocumentDetails
   - TaxDetails

   You will map elements from these complex types later in this lesson.
2. On the navigation page of the Customization Project Editor, open the Add SO303000 screen.
3. In the Commands area of the page, add containers and fields that correspond to the complex types and elements you discovered earlier, as shown in the following code.

```
add container "InvoiceSummary" {
  add field "Customer"
  add field "Date"
  add field "Type"
  add field "Status"
  add field "Description"
  add field "Amount"
  add field "Balance"
  add field "DiscountTotal"
  add field "GSTExemptTotal"
  add field "GSTTaxableTotal"
}
add container "DocumentDetails" {
  containerActionsToExpand = 1
  add field "Branch"
  add field "OrderNbr"
  add field "Warehouse"
  add field "Quantity"
  add field "UnitPrice"
  add field "Account"
  add field "Subaccount"
}
add container "TaxDetails" {
  add field "TaxID"
  add field "TaxRate"
  add field "TaxableAmount"
  add field "TaxAmount"
}
```

   The container corresponding to the form view of the form is called the primary container (in this case, InvoiceSummary), and the containers corresponding to form tabs are called secondary containers (in this case, DocumentDetails and TaxDetails). If a secondary container contains a grid, you can map it by adding a

corresponding container, as shown in the code above. Adding secondary containers with fields will be explored in the next lesson.

**Note:** The DocumentDetails container contains the containerActionsToExpand attribute. The attribute specifies how many container actions will be visible in the toolbar of the list view.

4. Save your changes and publish your project.
5. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu to make sure the Invoices screen is mapped.

## Step 1.1.2: Adding Standard Actions to the Invoices Screen

In this step, you will first add mandatory actions (Save and Cancel) and the Insert and Release actions to the Invoices screen (SO301000). To map these actions, do the following:

1. Learn the desired actions' titles in the WSDL schema. An example is shown in the following screenshot.

```
▼<s:complexType name="Actions">
 ▼<s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="CancelCloseToList" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="SaveCloseToList" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Save" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Cancel" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Insert" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Delete" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="CopyDocumentCopyPaste" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="PasteDocumentCopyPaste" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="SaveTemplateCopyPaste" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="First" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Previous" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Next" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Last" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="Release" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="VoidCheck" type="tns:Action"/>
    ...
    <s:element minOccurs="0" maxOccurs="1" name="CreditCCPayment" type="tns:Action"/>
    <s:element minOccurs="0" maxOccurs="1" name="SaveLoc" type="tns:Action"/>
  </s:sequence>
</s:complexType>
```

**Figure: The actions' names in the WSDL schema**

2. On the navigation page of the Customization Project Editor, open **the Add SO303000** screen.
3. In the **Commands** area of the page, add the actions to the InvoiceSummary container, as shown in the following code.

```
add container "InvoiceSummary" {
  # fields declaration
  ...
  add recordAction "Save" {
    behavior = Save
  }
  add recordAction "Cancel" {
    behavior = Cancel
  }
  add containerAction "Insert" {
    behavior = Create
  }
  add recordAction "ReleaseAction" {
    behavior = Record
  }
}
```

In the code above, you add actions and specify behavior for each action.

You will discover how to match an action on a form and its name in the WSDL schema in next lessons.

4. Save your changes and publish your project.
5. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu. Try to modify an invoice with "Balanced" status (because the **Release** action is available only for balanced invoices) and save your changes.

**Note:** The actions you mapped appear differently from one another on the mobile screen. The Save button appears when you modify something in an invoice record. The Cancel button corresponds to the Close Screen button. The Insert button appears in the list view of the Invoices screen. The Release button appears in the toolbar menu on the screen with an opened invoice that hasn't been released yet.

The Invoices screen (the list view and the form view) should look as shown in the following screenshots: the first screenshot shows the list view, the second screenshot shows the form view, and the third screenshot shows the form view the opened toolbar menu. The DocumentDetails and FinancialDetails containers are displayed as links at the bottom of the screen. You can open each of these containers by tapping the corresponding link.

**Note:** You cannot edit some of the fields because modifying them is not yet supported by MYOB Advanced Mobile Framework.

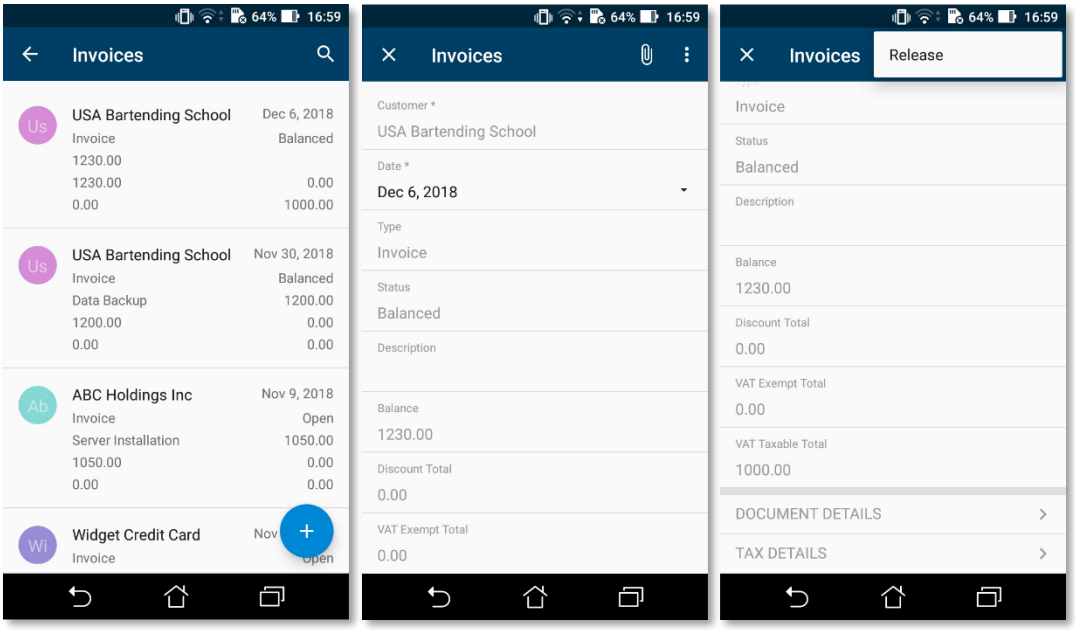


**Figure: The Invoices screen**

## Lesson Summary

In this lesson, you have added a number of essential fields to the Invoices screen (SO301000) and learned how to map different types of actions.

# Lesson 1.2: Configuring the Layout of the Screen

By default, all fields mapped to the mobile screen are displayed one after another. It may not be very convenient, however, for the user to see all of the fields on the mobile device. MSDL offers the functionality to organize the fields on a screen in a more user-friendly way.

In this lesson, you will organize the fields and containers of the Invoices screen (SO303000). First you will create a header for the screen, then you will put several fields in a group, and finally you will organize different kinds of tabs on the screen.

## Step 1.2.1: Organizing the Fields of the InvoiceSummary Container

To organize multiple fields of one container, you can use a layout object and define the fields within the object. To display the fields of the InvoiceSummary container in a header, do the following:

1. On the Mobile Application page of the Customization Project Editor, open the **Add SO303000** screen.
2. In the **Commands** area, add the following code instead of the field definitions of the InvoiceSummary container.

```
add layout "InvoiceHeader" {
  layout = "HeaderSimple"
  add layout "InvoiceHeaderRow1" {
    layout = "Inline"
    add field "Type"
    add field "Date"
  }
  add layout "InvoiceHeaderRow2" {
    layout = "Inline"
    add field "Status"
    add field "Amount"
  }
  add layout "InvoiceHeaderRow3" {
    layout = "Inline"
    add field "Customer"
  }
  add layout "InvoiceHeaderRow4" {
    layout = "Inline"
    add field "Description"
  }
}
```

   Note that several fields you mapped before have been left outside of the header. You will organize them in the next step.

   In this code, you first define a layout object of a HeaderSimple type (called InvoiceHeader), and then in the InvoiceHeader layout object, you define layout objects of the Inline type that correspond to rows. In each layout object of the Inline type, you add the fields that you want to be displayed in the same row.

3. Save your changes, and publish your project.
4. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu. Check the fields of the InvoiceSummary container. They should look as shown in the following screenshot.


**Figure: The header of the Invoice screen**

## Step 1.2.2: Grouping Fields

You can put multiple fields into an expanding group, regardless of which container the fields originate from. To explore this functionality, you will add fields from the FinancialDetials container and configure a group for them. Do the following:

1. On the Mobile Application page of the Customization Project Editor, open the **Add SO303000** screen.
2. In the **Commands** area, add the fields of the FinancialDetails container to theInvoiceSummary container, as shown below.

```
add field "FinancialDetails#ARAccount"
add field "FinancialDetails#Branch"
add field "FinancialDetails#ARSubaccount"
```

   As you see, you can access the fields of another container by composing a string consisting of the container's name, the # sign, and the field name.

3. Add a group object, and declare the fields within it.

```
add group "FinancialDetails" {
  displayName = "Financial Details"
  collapsable = True
  collapsed = True
  add field "FinancialDetails#ARAccount"
  add field "FinancialDetails#Branch"
  add field "FinancialDetails#ARSubaccount"
}
```

4. This group should be able to fold and unfold and is folded by default.
5. Save your changes, and publish your project.
6. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu. Verify that the Financial Details group appears, and open it. It should look as shown in the following screenshot. By default, this group is collapsed because you used the collapsed = True attribute.

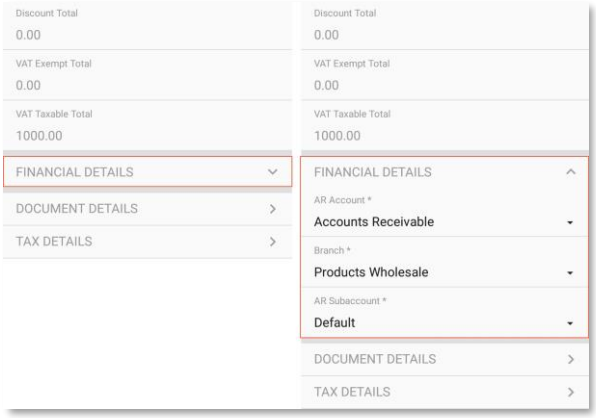


**Figure: The Financial Details group of the Invoices screen**

## Step 1.2.3: Organizing the Secondary Containers with Grids of the Invoices Screen

If you have multiple fields or containers on a screen, you can display them as tabs. Do the following to organize the previously defined fields and containers into tabs:

1. On the Mobile Application page of the Customization Project Editor, open the Add Invoices screen.
2. In the Commands area, add the following code to the InvoiceSummary container.

```
add layout "Other" {
  layout = Tab
  add field "GSTExemptTotal"
  add field "GSTTaxableTotal"
}

add layout "DocDetails" {
  layout = DataTab
  add containerLink "DocumentDetails" {
    control = "ListItem"
  }
}

add layout "TaxDetails" {
  layout = DataTab
  add containerLink "TaxDetails" {
    control = "ListItem"
  }
}
```

This code unites two fields of the InvoiceSummary container into one tab and links to two previously defined containers into two other tabs.

**Note:** The tabs that contain links to containers should be of the DataTab type.

3. Save your changes and publish your project.
4. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu. On this screen, the added tabs should look as shown in the following screenshot.

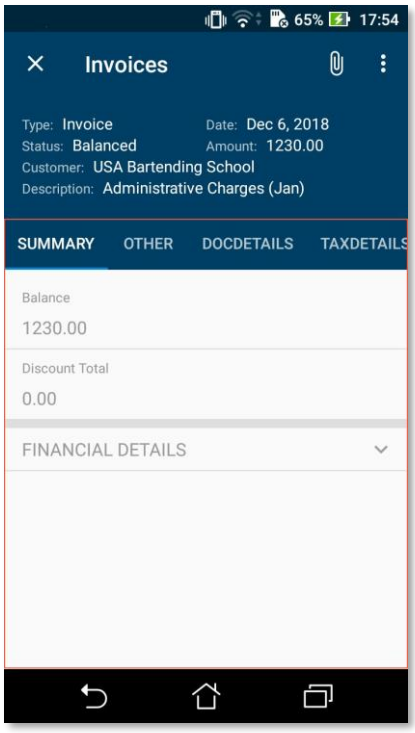


**Figure: The tabs of the Invoice screen**

**Note:** All fields and groups that were not placed in the header layout or tabs are placed in the Summary tab that is created automatically.

## Lesson Summary

In this lesson, you have learned how to configure different types of layouts on a mobile app screen.

# Lesson 1.3: Configuring Related Containers

In previous lessons, you have already learned the following ways of using containers that are related to one another:

- Declaring a secondary container and creating a link to it within a data tab (Step 1.2.3)
- Displaying fields from one container in another container (Step 1.2.2)

In this lesson, you will learn how to display information from many-to-one containers: containers that look like dialog boxes in the browser version of MYOB Advanced.

## Configuring a Screen with Many-to-One Containers

The browser version of MYOB Advanced includes dialog boxes that are opened from forms. These dialog boxes often provide additional actions for items in a grid. On the Invoices form (SO303000), for example, you could use the **Add Order** dialog box to add sales orders to the invoice. (See the following screenshot.)

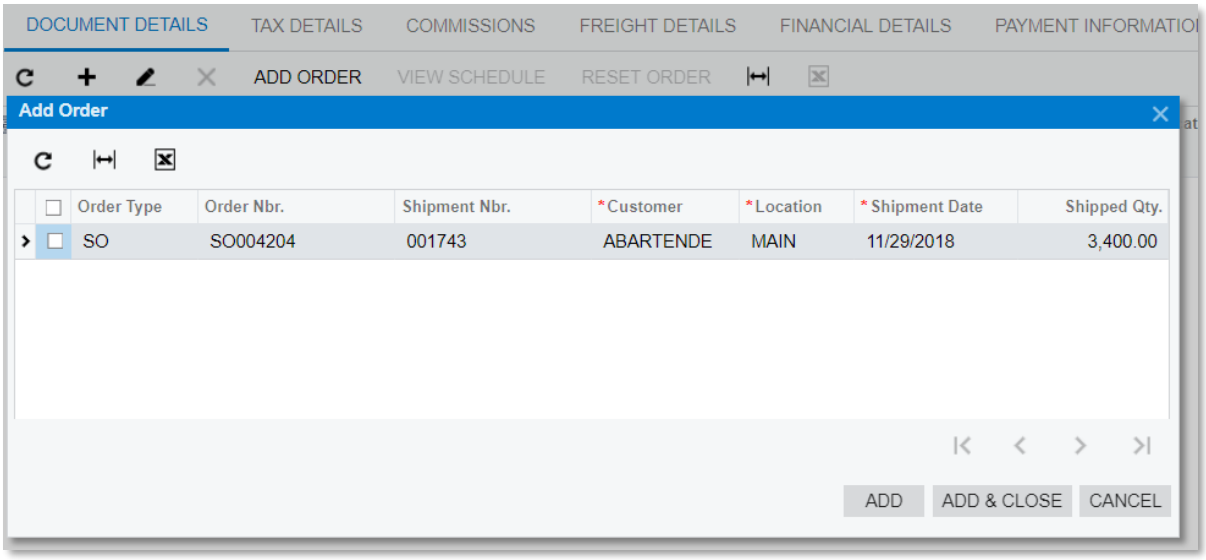


**Figure: The Add Order dialog box of the Document Details tab**

To use the **Add Order** dialog box in the mobile app, you should map the **Add Order** dialog box as follows.

1. Find the **Add Order** dialog box description in the WSDL schema, as shown in the following screenshot.

```
▼<s:complexType name="AddOrder">
  ▼<s:sequence>
     <s:element minOccurs="0" maxOccurs="1" name="DisplayName" type="s:string"/>
     <s:element minOccurs="0" maxOccurs="1" name="Selected" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="OrderType" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="OrderNbr" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="ShipmentNbr" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="Customer" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="Location" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="ShipmentDate" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="ShippedQty" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="ServiceCommands" type="tns:AddOrderServiceCommands"/>
  </s:sequence>
</s:complexType>
```

**Figure: The Add Order dialog box description**

2. Find the **Add Order** action name in the WSDL schema as follows:
    a. On the form title bar of the Invoices form, click **Customization > Inspect Element**.
    b. On the toolbar of the Document Details tab, click the **Add Order** action.
    c. When the Element Properties dialog box is opened, note the **Action Name** field value, and click **Actions > View Aspx Source**.
    d. In the Source Code window that opens, on the **Screen ASPX** tab, find the action name you noted in the previous substep.
    e. Within the tag with a key name equal to the desired action name, find the AutoCallBack tag.

    The value of the Command attribute is the name of the action in the WSDL schema. The following screenshot shows the ASPX code for the **Add Order** action.

```
<px:PXToolBarButton Text="Add Order" Key="cmdShipmentList">
    <AutoCallBack Command="SelectShipment" Target="ds">
        <Behavior PostData="Page" CommitChanges="True" ></Behavior>
    </AutoCallBack>
</px:PXToolBarButton>
```

**Figure: The ASPX code of the Add Order button**

    As you can see in the WSDL schema in the Action complex type, the name of the **Add Order** action is SelectShipment. You can discover another way to learn the WSDL name of an object in the next part of the training document. For details, see "Lesson 2.1: Finding Object Names in the WSDL Schema".

3. On the Mobile Application page of the Customization Project Editor, open the Add Invoices screen.
4. In the Commands area, add the AddOrder container as shown below.

```
add container "AddOrder" {
  type = SelectionActionList
  visible = False
  listActionsToExpand = 1
  add field "Selected"
  add field "OrderType"
  add field "OrderNbr"
  add field "Location"
  add listAction "AddShipment" {
    icon = "system://Check"
    behavior = Void
  }
}
```

    You can again find the name for the list action that adds orders in the dialog box in the ASPX source code of the page.

5. In the DocumentDetails container, add the action that opens the AddOrder container in a separate mobile screen, as shown in the following code.

```
add containerAction "SelectShipment" {
behavior = Void
redirect = True
redirectToContainer = "AddOrder$List"
}
```

    The name of the action should be that one you found previously in the ASPX code. The AddOrder$List structure means that the AddOrder container should be opened as a list.

6. Save your changes, and publish your customization project.
7. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu.
8. Open an invoice which has not been released so that you can update it. Open the DocDetails tab. On the screen toolbar, open the menu, and tap Add Order. In the screen that opens, select an order and save your changes. After returning to the Invoices screen you can see the added order and save the invoice. See the following screenshots; the first screenshot shows the DocDetails tab with its menu, the second shows the Add Order screen, the third and the fourth show the added order before and after saving. In Lesson 8, you will learn more about mapping and setting up this type of screen.

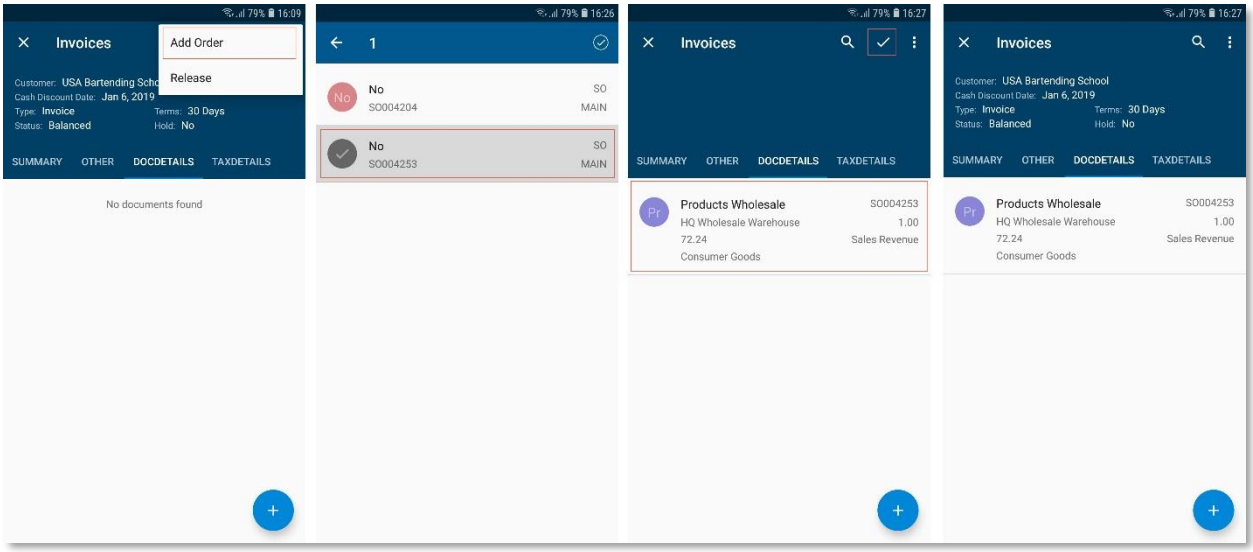


**Figure: Opening the Add Order screen**

## Lesson Summary

In this lesson, you have learned how to configure a detail container opened from the master container.

# Part 2: Redirecting the User to Different Containers and Screens

You can redirect the user to different screens and containers in the mobile app in one of the following ways:

- Allow a redirection that is already implemented in MYOB Advanced
- Create a new redirection

You can redirect the user to the following objects:

- A container within the current screen. You have learned about this option in Lessons 1.2 and 1.3.
- Another screen. This redirection can be implemented by using form customization and is described in Lesson 3.1.

Redirection to a secondary container within another screen is impossible.

## Lesson 2.1: Finding Object Names in the WSDL Schema

To map an object (a container, a field, or an action), you have to know its name in the WSDL schema of the form. The way you determine the object's name depends on the type of the object.

### Step 2.1.1: Determining a Container's Name

Names of all the form's containers are listed in the Content complex type of the WSDL schema. The primary container of a form view usually has the following naming convention: "<FormTitle>Summary". Sometimes the singular version of the form name is used in the container name instead of the plural version used in the actual name. For example, for the Invoices (SO303000) form, the primary container is named "InvoiceSummary" in the WSDL schema. Sometimes the name of the primary container corresponds to the form name exactly, as it does for the Expense Receipts (EP301010) form, where the primary container's name is "ExpenseReceipts".

A tab, which is a secondary container, usually has a name corresponding to the tab's name.

For example, on the Invoices form, the container for the Document Details tab is named "DocumentDetails", the container for the Tax Details tab is named "TaxDetails", and the container for the Commissions tab is named Commissions.

To find the name of container that corresponds to the Payment Information tab of the Invoices form, do the following:

1. Open the WSDL schema of the Invoices form.
2. In the Content complex type, find the element (container name) that corresponds to the Payment Information tab. As it was described above, the name of the container is the same as tha tab's name: "PaymentInformation".
3. In the WSDL schema, find the "PaymentInformation" complex type. It contains fields that you can map in the PaymentInformation container.

For a dialog box (which is also a secondary container) that opens when you select an action on a toolbar, the container usually has the same name as the dialog box. For example, the container of the Add Order dialog box (which is invoked on the Document Details tab of the Invoices form) is named "AddOrder".

## Step 2.1.2: Determining a Field's Name

The field name in the WSDL schema usually corresponds to the name of the field in the user interface. But first you should determine which container the field is in, because the same field name can exist in multiple containers.

To find the field name of the Date user interface element on the Invoices (SO303000) form, do the following:

1. Open the WSDL schema of the Invoices form.
2. Determine the name of the primary container of the Invoices form in the WSDL schema. Because you know that the Date UI element is located in the primary container of the form, you would need to first determine the name of the primary container of the Invoices form. As was determined in Step 4.1, the name of the primary container is "InvoiceSummary".
3. In the InvoiceSummary complex type of the WSDL schema, find the field name of the Date element in the WSDL schema, as shown in the screenshot below.


**Figure: The Date field of the Invoices form in the WSDL schema**

## Step 2.1.3: Determining an Action's Name

All the form's action are listed in the Actions complex type of the WSDL schema. You can find the action's name there or learn it from the action's properties as described below.

To learn the name of an action on a form toolbar or a grid (tab) toolbar, you need to find the Command property of the action in the Layout Editor. For example, to find the name of the Add Order action on the table toolbar of the Document Details tab of the Invoices (SO303000) form, do the following:

1. Open the Invoices form.
2. On the form title bar, click **Customization > Inspect Element**.
3. On the table toolbar of the Document Details tab, click the **Add Order** button.
4. In the Element Properties dialog box, which opens, click **Customize**. In the dialog box that opens, select the customization project in which you customize the mobile app.
5. The Layout Editor of the Customization Project Editor opens for the form.
6. In the control tree pane (the left pane), select **Tab > Document Details > Grid: Transactions > ActionBar-CustomItems > Add Order**.

7. On the Properties tab of the Add Order action, find the **Base Properties** node and the open the **AutoCallBack** node inside it.
8. Find the **Command** property in the list. Its value is the name of the action in the WSDL schema. In this case, the name of the **Add Order** action is selectShipment (see the following screenshot).

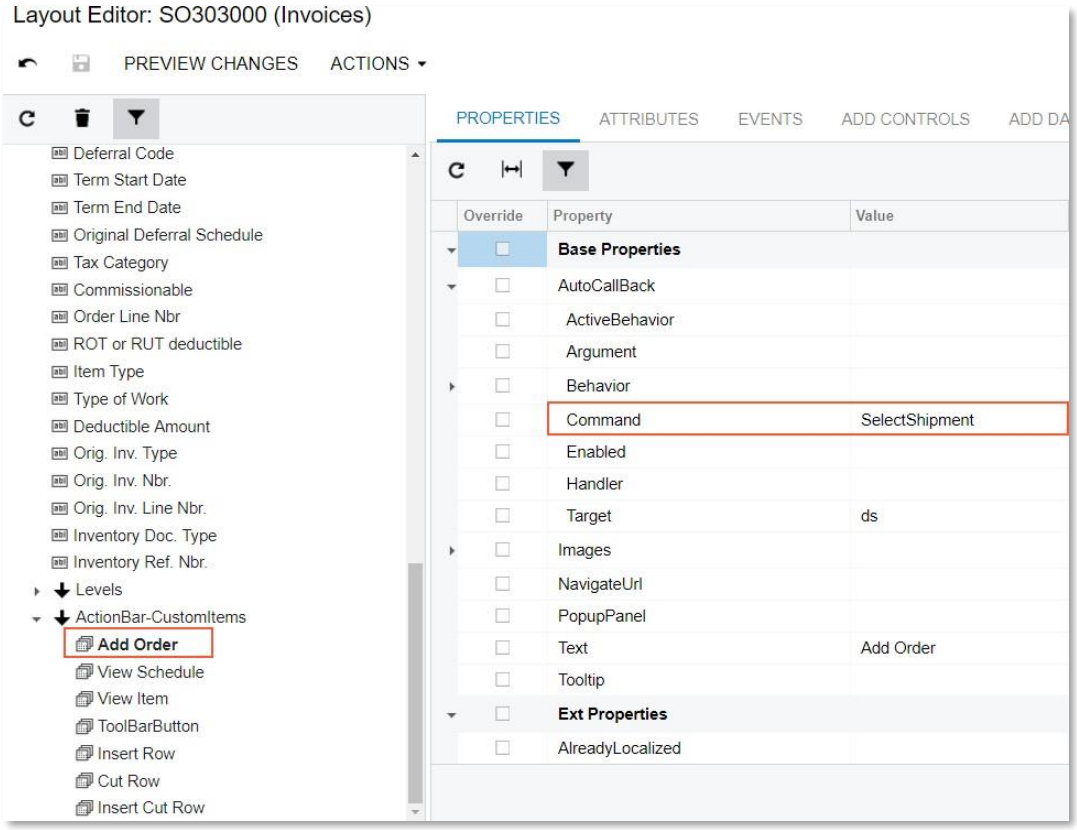


**Figure: The name of the Add Order action**

In some cases, the action's name may be stored in the CommandName property of the **Ext Properties** node.

## Self-Guided Exercise: Finding Object Names for Further Mapping

In Lesson 5, you will map the primary containers of the Customers form (AR303000) and the Customer Location form (AR303020), and the **Add Location** action on the Customers form. Find the object names corresponding to the primary containers of the Customers form and Customer Location form, and the object name for the Add Location action on the Customers form, in the WSDL schema by using the methods described above.

## Lesson Summary

In this lesson, you have learned how to determine the name of objects (that is, containers, fields, and actions) in the WSDL schema.

# Lesson 2.2: Making Use of a Redirection Implemented in MYOB Advanced

On the Customers form (AR303000) of MYOB Advanced, you can open the Customer Locations form (AR303020) by clicking the Add Location button on the table toolbar of the Locations tab. In this lesson, you will learn how to map this functionality to the mobile app.

First, you will map the Customer and Customer Locations forms as self-guided exercises. Then you will map an action that redirects the user from one form to another by using the logic of the Customers form.

## Step 2.2.1: Mapping the Customers Form (Self-Guided)

In this step, you should do the basic mapping of the Customers form (AR303000): Add the Customers screen to the mobile site map and to the app's main menu. The Customers screen should include all of the following: the CustomerSummary container with some essential fields, the Locations container, and the Save and Cancel actions. An example of the mapped screen is shown below.

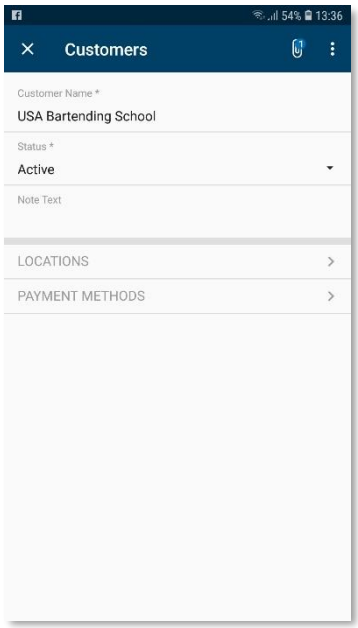


**Figure: The Customers screen**

## Step 2.2.2: Mapping the Locations Form (Self-Guided)

In this step, you should do the basic mapping of the Customer Locations (AR303020) form: Add the Customer Locations screen to the mobile site map and to the app's main menu. Note that on the main menu, the Customer Locations item should have the visible attribute set to False because this screen should be opened only from the Customers screen. The Customer Locations screen should include the LocationSummary container with some essential fields, and the standard Save and Cancel actions. The sample code for the screen is presented below.

```
add screen AR303020 {
  openAs = Form
  add container "LocationSummary" {
    add field "Customer"
    add field "LocationID" {
      PickerType = Searchable #to be able to add ID manually
    }
    add field "Active"
    add field "LocationName"
    add field "NoteText"
    add recordAction "Save" {
      behavior = Save
    }
    add recordAction "Cancel" {
      behavior = Cancel
    }
  }
}
```

## Step 2.2.3: Declaring the Redirect Action

After you have mapped the Customers (AR303000) and Customer Locations (AR303020) forms, you should declare an action that redirects a user from the Customer screen to the Customer Locations screen as follows:

1. On the Locations tab of the Customers form, find the **Add Location** button. Learn the name of this action in the WSDL schema by inspecting the element. If you completed the self-guided exercise in the previous lesson, you determined the **Add Location** action's name in the WSDL schema by using the Layout Editor.

**Note:** In some cases, you can do the following instead: Invoke **Inspect Element**, and notice the action's name in the **Action Name** element of the Element Properties dialog box, as shown in the following screenshot.

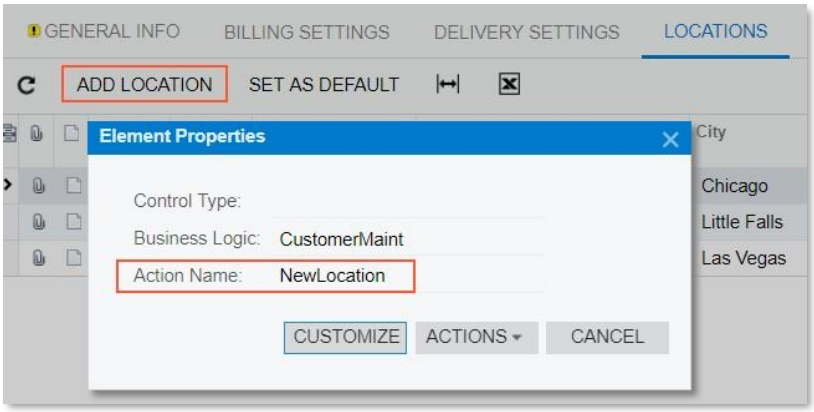


**Figure: The Add Location action name**

2. On the Mobile Application page of the Customization Project Editor, open the Add: AR303000 Customers page, add the action as shown in the following code.

```
add recordAction "NewLocation" {
  displayName = "Add Location"
  behavior = Record
  redirect = True
}
```

You do not need to specify to which screen the action will redirect the user, because it is already implemented in the action's default business logic.

3. Save your changes, and publish your customization project.

4. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Customer screen on the main menu. Select any customer, and on the screen's menu, tap **Add Location**. The Customer Locations screen opens. The first screenshot below shows the **Add Location** button on the screen toolbar, the second screenshot shows the screen where you add the new location, and the third screenshot shows entered location information with the **Save** button.

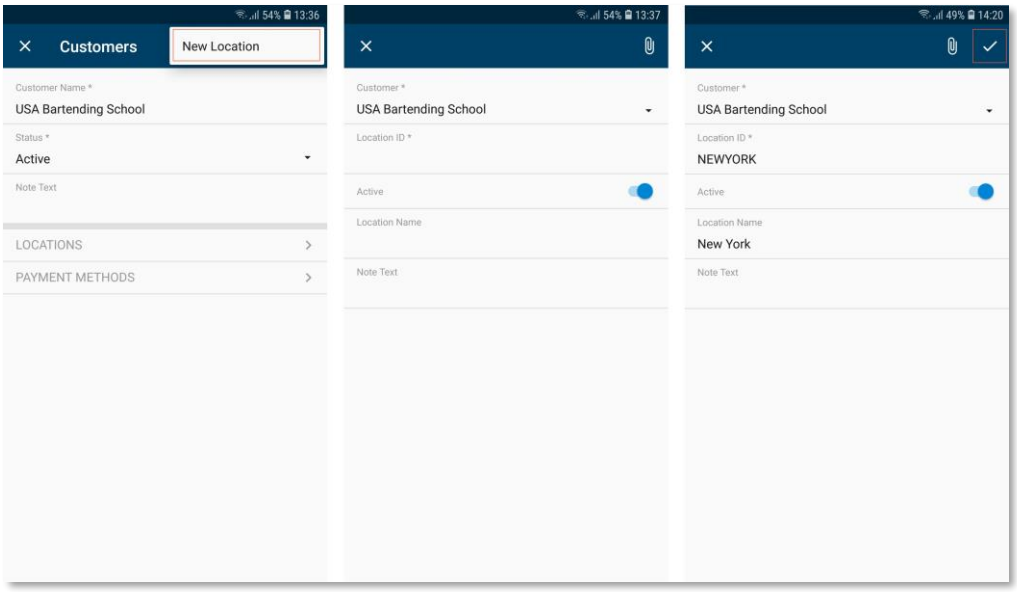


**Figure: Adding a new customer location**

After you added the new location, you can see it in the Location screen that you open from the Customer screen as shown on screenshots below.

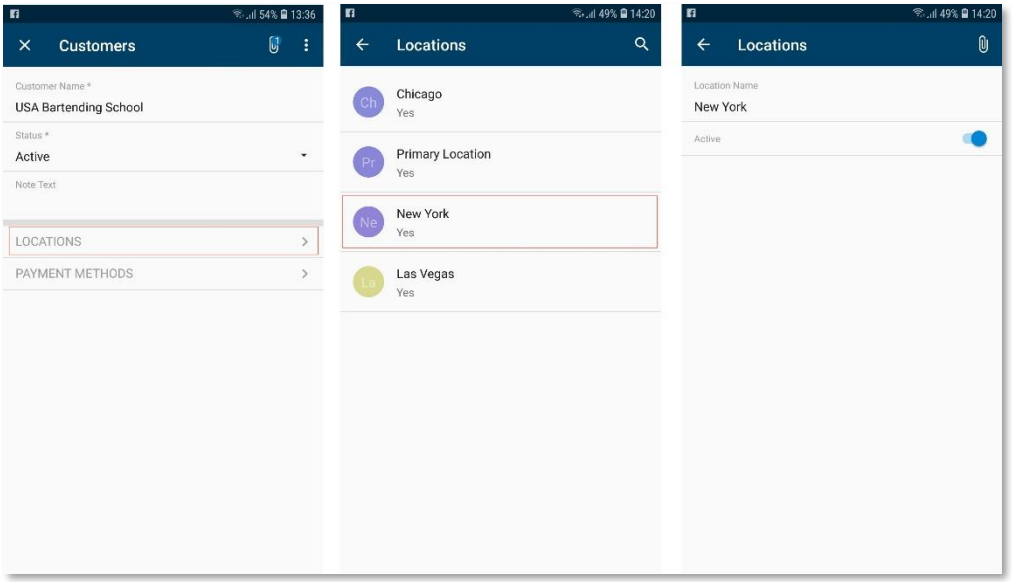


**Figure: Checking the added location**

## Lesson Summary

In this lesson, you have learned how to map in the mobile app redirection from one screen to another that is already implemented in MYOB Advanced.

# Part 3: Advanced Use Cases

This part contains use cases that demonstrate solving some common issues when you are customizing the MYOB Advanced mobile app.

Lesson 3.1 of this part requires experience in creating customizations of MYOB Advanced forms. For details, see the Customization Guide.

# Lesson 3.1: Adding a URL Link on a Mobile App Screen

It is impossible to add a URL link on a mobile app screen by using only MSDL unless the link already exists on the corresponding MYOB Advanced form because MSDL only maps an existing functionality of a form. So, to have a link on a mobile app screen, you should first customize an MYOB Advanced form by adding a new action, and then map this action to the mobile screen.

In this lesson, you will customize the Invoices form (SO303000) by adding a toolbar button with a hyperlink, and then you will map this button to the mobile screen.

## Step 3.1.1: Adding a New Toolbar Button to the Invoices Form

To add a new toolbar button to the Invoices form (SO303000) of MYOB Advanced, do the following:

1. Add a new action to the Invoices form. The code should look as follows.

```
public PXAction<PX.Objects.AR.ARInvoice> TestURL;

[PXButton(CommitChanges = true)]
[PXUIField(DisplayName = "TestURL")] protected void testURL()
{
      // the body of the action delegate method
}
```

2. In the action delegate method, add the following code, which redirects the user to the external URL.

```
throw new PXRedirectToUrlException("http://www.myob.com",
                                   "Redirect:http://www.myob.com");
```

   This code instructs the system to open http://www.myob.com in a new tab of the default browser. The code uses an exception instead of usual methods like System.Diagnostics.Process.Start(), because web applications are normally forbidden to run these processes as opening external URLs.

3. Save your changes and publish your customization project.

4. Open the Invoices form, and make sure that a new action has appeared on the form toolbar, as shown on the following screenshot.

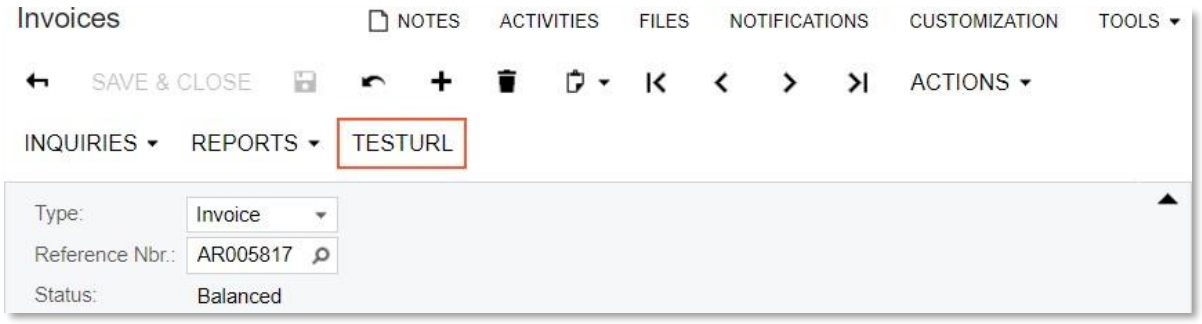


**Figure: The new action of the Invoices form**

## Step 3.1.2: Mapping the New Action to the Mobile App

Now that you have created the new TestURL action on the Invoices form (SO303000) toolbar, you can map it to the Invoices screen as follows:

1. Open the WSDL schema of the customized Invoices form. Find the Actions complex type and the new TestURL action, as shown in the following screenshot.

```
<s:element minOccurs="0" maxOccurs="1" name="CaptureCCPayment" type="tns:Action"/>
<s:element minOccurs="0" maxOccurs="1" name="AuthorizeCCPayment" type="tns:Action"/>
<s:element minOccurs="0" maxOccurs="1" name="VoidCCPayment" type="tns:Action"/>
<s:element minOccurs="0" maxOccurs="1" name="CreditCCPayment" type="tns:Action"/>
<s:element minOccurs="0" maxOccurs="1" name="TestURL" type="tns:Action"/>
<s:element minOccurs="0" maxOccurs="1" name="SaveLoc" type="tns:Action"/>
```

**Figure: The TestURL action in the WSDL schema**

2. On the Mobile Application page of the Customization Project Editor, open the Add Invoices screen.
3. In the Commands area, add the TestURL action, as shown below.

```
add recordAction "TestURL" {
  behavior = Void
  redirect = True
}
```

4. Save your changes and publish your customization project.

5. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Invoices screen on the main menu. On the screen toolbar, open the menu, and tap TestURL. The built-in browser with the MYOB Advanced main page opens. The TestURL button on the toolbar menu and the opened webpage are shown on the screenshots below.

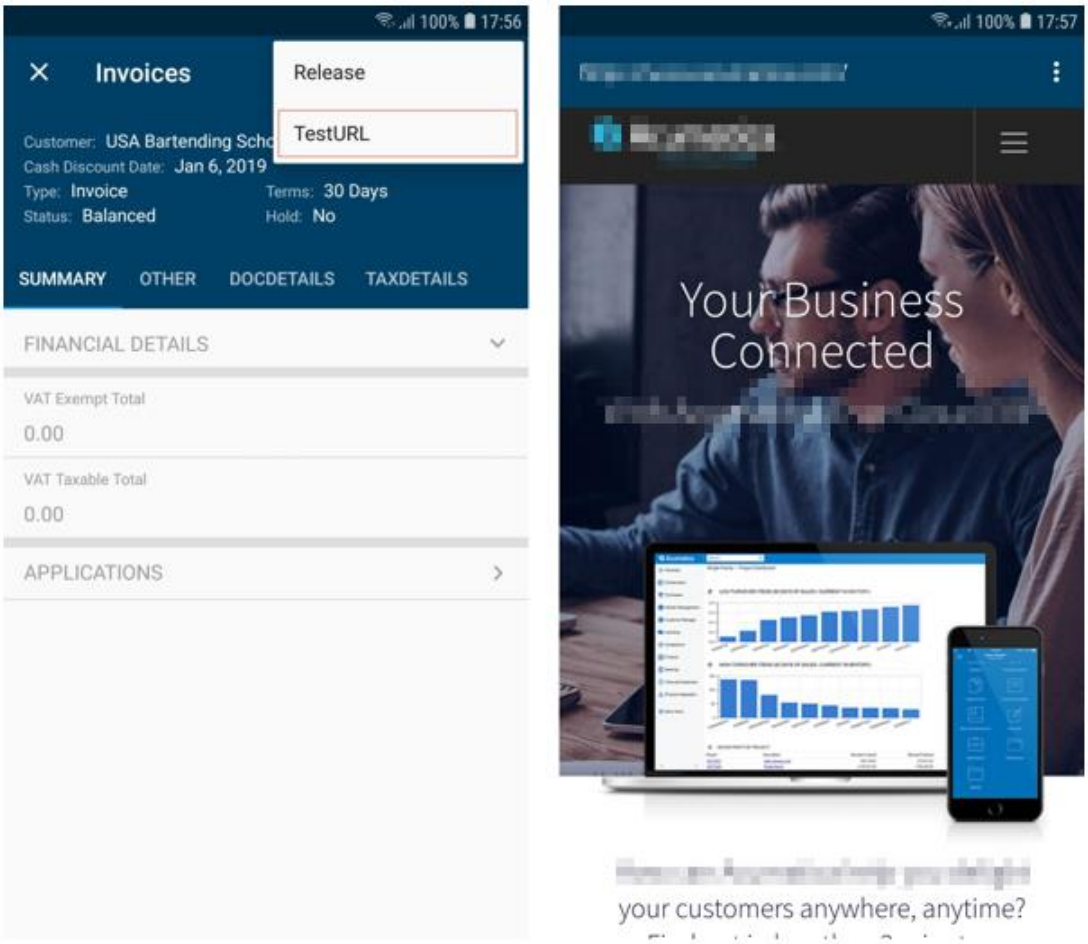


**Figure: A URL link in the mobile app**

## Lesson Summary

In this lesson, you have learned how to add to a mobile app screen a link to an external URL.

# Lesson 3.2: Implementing Multiple Redirections

Sometimes when multiple complex redirections are implemented in the mobile app, the Back button does not redirect the user properly to the previous screen. This can happen if redirections were not done correctly.

This situation will be illustrated with the example of adding purchase order lines (PO lines) to a purchase receipt on the Purchase Receipts form (PO302000).

## Step 3.2.1: Analysing the Use Case in the Browser Version of MYOB Advanced

To understand what you should map to the mobile app, you should first analyse the use case of adding any number of purchase order lines to a purchase receipt in MYOB Advanced. Do the following:

1. Make sure at least one purchase order with the Open status exists in the system. If this is not the case, create one on the Purchase Orders (PO301000) form.
2. On the Purchase Receipts (PO302000) form, open a receipt.
3. On the table toolbar of the Document Details tab, click **Add PO Line**. The Add Purchase Order Line dialog box, which contains the list of open purchase orders, opens.
4. In the dialog box, select the Included check box for each purchase order line you want to add, and click **Add & Close**. The dialog box is closed, and the selected purchase order lines are added on the Document Details tab.

Based on the use case in which purchase order lines are added to a purchase receipt, in the mobile site map, you have to update the Purchase Receipts screen. (You don't have to create it from scratch because it is already mapped by default.) In the update, you should map the **Add PO Line** action and the Add Purchase Order Line dialog box with the actions of this dialog box.

## Step 3.2.2: Updating the Purchase Order Screen

Now you will map the actions and the dialog box explained in the previous step as follows:

1. Open and analyse the WSDL schema of the Purchase Receipts form (PO302000). Find elements corresponding to the **Add PO Line** action and Add Purchase Order Line dialog box, as shown in the following screenshot.

**Note:** You can use methods of determining WSDL schema names of form elements presented in Lesson 4.

```
▼<s:complexType name="Actions">
  ▼<s:sequence>
     <s:element minOccurs="0" maxOccurs="1" name="CancelCloseToList" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="SaveCloseToList" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="Save" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="Cancel" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="Insert" type="tns:Action"/>
```

```
     <s:element minOccurs="0" maxOccurs="1" name="Assign" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOOrder" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOOrder2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddTransfer" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddTransfer2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddINTran" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddINTran2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOOrderLine" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOOrderLine2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOReceiptLine" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOReceiptLine2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOReceiptReturn" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOReceiptReturn2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOReceiptLineReturn" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="AddPOReceiptLineReturn2" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="ViewPOOrder" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="CreateAPDocument" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="CreateLCDocument" type="tns:Action"/>
     <s:element minOccurs="0" maxOccurs="1" name="SaveLoc" type="tns:Action"/>
   </s:sequence>
 </s:complexType>
```

```
▼<s:complexType name="AddPurchaseOrderLine">
  ▼<s:sequence>
     <s:element minOccurs="0" maxOccurs="1" name="DisplayName" type="s:string"/>
     <s:element minOccurs="0" maxOccurs="1" name="Selected" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="OrderNbr" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="Vendor" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="LineType" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="InventoryID" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="Subitem" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="UOM" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="OrderQty" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="QtyOnReceipts" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="OpenQty" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="LineDescription" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="PromisedDate" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="MinReceipt" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="MaxReceipt" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="ReceiptAction" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="LineNbr" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="NoteText" type="tns:Field"/>
     <s:element minOccurs="0" maxOccurs="1" name="ServiceCommands" type="tns:AddPurchaseOrderLineServiceCommands"/>
   </s:sequence>
 </s:complexType>
```

**Figure: The Add PO Line action and the Add Purchase Order Line dialog box**

2. In the Customization Project Editor, open the Update: PO302000 Purchase Receipts page.
3. Analyse the mapped objects in the Result Preview area.
4. Add the **Add PO Line** action, which opens the Add Purchase Order Line dialog box.

   Note that in the WSDL schema there are two actions: AddPOOrderLine and AddPOOrderLine2. The first one opens the AddPOOrderLine dialog box, and the second one saves the added lines and closes the dialog box (the **Add & Close** button in the browser version).

   So, to add the **Add PO Line** action, which opens the Add Purchase Order Line dialog box, you should map the AddPOOrderLine. If you want it to appear on the primary container of the Purchase Receipts screen (that is, the DocumentSummary container), add a record action, as shown below.

```
update screen PO302000 {
  update container "DocumentSummary" {
    add recordAction "AddPOOrderLine" {
      displayName = "Add PO Line"
      behavior = Void
      redirect = True
      redirectToContainer = "AddPurchaseOrderLine$List"
    }
  }
}
```

If you want the **Add PO Line** action to appear in the DocumentDetails container, to correspond with the UI of the browser version (in the browser version of the Purchase Receipts form, the **Add PO Line** button is on the Document Details tab), add a container action in the DocumentDetails container as shown below.

```
update container "DocumentDetails" {
  add containerAction "AddPOOrderLine" {
    Behavior=Void
    Redirect=true
    RedirectToContainer="AddPurchaseOrderLine$List"
  }
}
```

5. Map the Add Purchase Order Line dialog box by adding the AddPurchaseOrderLine container to the Purchase Receipts screen. Within the container, declare the AddPOOrderLine2 action, which adds the selected lines to the DocumentDetails container, as shown in the code below. You could have seen the AddPOOrderLine2 action in the WSDL schema presented in the figure above.

```
add container "AddPurchaseOrderLine" {
  Type = SelectionActionList
  Visible = false

  add field "Selected"
  add field "OrderNbr"
  add field "InventoryID"
  add field "LineDescription"
  add field "OrderQty"

  add listAction "AddPOOrderLine2" {
    Behavior = Void
    displayName = "Add & Close"
  }
}
```

## Step 3.2.3: Testing the Use Case in the Mobile App

After you mapped the needed containers and actions, you can test the redirection use case on a mobile device as follows:

1. Open the MYOB Advanced mobile app, sign in to your instance, and navigate to the Purchase Receipts screen.
2. In the Purchase Receipts screen, scroll to the Document Details container and open it.
3. In the Document Details screen, open the toolbar menu and tap **Add PO Line**. The Add PO Line screen opens. The title of the screen shows the number of selected lines.
4. In the Add PO Line screen, select a line you want to add to the purchase receipt.
5. Open the toolbar menu and tap **Add & Close**. The app returns to the Document Details screen with selected lines added to the list.

6. If needed, select the added line in the list and modify the receipt quantity so that it is more than zero. To save changes in the Document Details screen, tap the Back button on the screen toolbar.
7. When you tap Back on the Document Details screen toolbar, the app returns to the Purchase Receipts screen. The added lines are not yet saved so the Save button appears on the toolbar.
8. On the Purchase Receipts screen, tap the Save button. The purchase receipt is saved.

You can see described screens on the figure below from left to right from top to bottom.

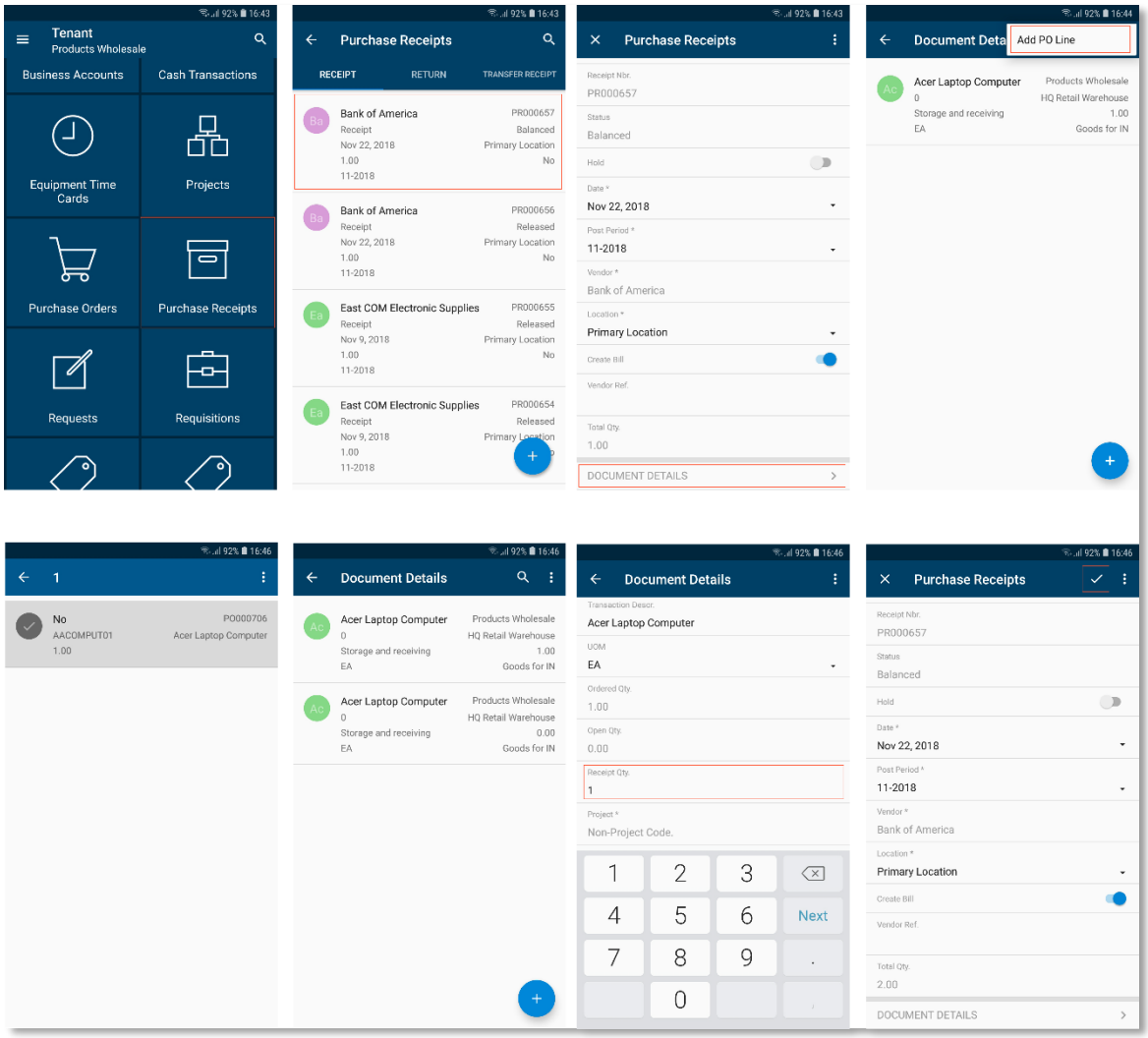


**Figure: Adding a PO line to a purchase receipt**

## Lesson Summary

In this lesson, you have learned how to correctly map and configure multiple redirects to related containers.