



vuforia[™] studio

Metadata 303

**Creating a Persistent Cart Using
ThingWorx**

Copyright © 2020 PTC Inc. and/or Its Subsidiary Companies. All Rights Reserved.

User and training guides and related documentation from PTC Inc. and its subsidiary companies (collectively "PTC") are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed software user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. Training materials may not be copied without the express written consent of PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes. Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC.

UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.

PTC regards software piracy as the crime it is, and we view offenders accordingly. We do not tolerate the piracy of PTC software products, and we pursue (both civilly and criminally) those who do so using all legal means available, including public and private surveillance resources. As part of these efforts, PTC uses data monitoring and scouring technologies to obtain and transmit data on users of illegal copies of our software. This data collection is not performed on users of legally licensed software from PTC and its authorized distributors. If you are using an illegal copy of our software and do not consent to the collection and transmission of such data (including to the United States), cease using the illegal version, and contact PTC to obtain a legally licensed copy.

Important Copyright, Trademark, Patent, and Licensing Information:

See the About Box, or copyright notice, of your PTC software.

UNITED STATES GOVERNMENT RIGHTS

PTC software products and software documentation are “commercial items” as that term is defined at 48 C.F.

R. 2.101. Pursuant to Federal Acquisition Regulation (FAR) 12.212 (a)-(b) (Computer Software) (MAY 2014) for civilian agencies or the Defense Federal Acquisition Regulation Supplement (DFARS) at 227.7202-1(a) (Policy) and 227.7202-3 (a) (Rights in commercial computer software or commercial computer software documentation) (FEB 2014) for the Department of Defense, PTC software products and software documentation are provided to the U.S. Government under the PTC commercial license agreement. Use, duplication or disclosure by the U.S. Government is subject solely to the terms and conditions set forth in the applicable PTC software license agreement.

PTC Inc., 121 Seaport Blvd, Boston, MA 02210 USA

Prerequisites

Completion of the following tutorials:

Metadata 101 – Using Attributes in Creo Illustrate

Metadata 201 – Using JavaScript to Highlight Parts and Create Ionic Popups

Metadata 202 – Using JavaScript to Find Parts

Metadata 301 – Adding Pricing Data and a Shopping Cart to a Model

Metadata 302 – Add a Simple ThingWorx Service to Vuforia Studio

Intro

Think of buying something on the internet; most of the time, when you add something to your cart it will stay there until you check out or remove the object. This is the concept of a persistent shopping cart. Up until now, when you had added an item into your cart in your Vuforia Studio experience, it disappeared whenever you restarted your experience. This issue can be resolved with the help of ThingWorx being connected to Vuforia Studio to store part information in a persistent cart, which will be the focus of this exercise.

In order to complete this exercise, you must have completed each part of the Metadata series of projects before this.

The following topics will be covered in this project. Jump to them with their hyperlinks:

[Metadata 303.1 Becoming Familiar with cartThing](#)

[Metadata 303.2 Add cartThing to Vuforia Studio](#)

[Metadata 303.3 Binding cartThing and Editing the 2D Canvas](#)

[Metadata 303.4 Invoking the addToCart Service, Accounting for Welded Parts, and Cleaning Up the Code](#)

[Metadata 303.5 Using the Experience](#)

There is also an [appendix](#) at the end of the document for the completed code of this project.

All important notes and UI areas are **Bold**.

All non-code text to be typed is *italicized*.

All code follows `this convention`

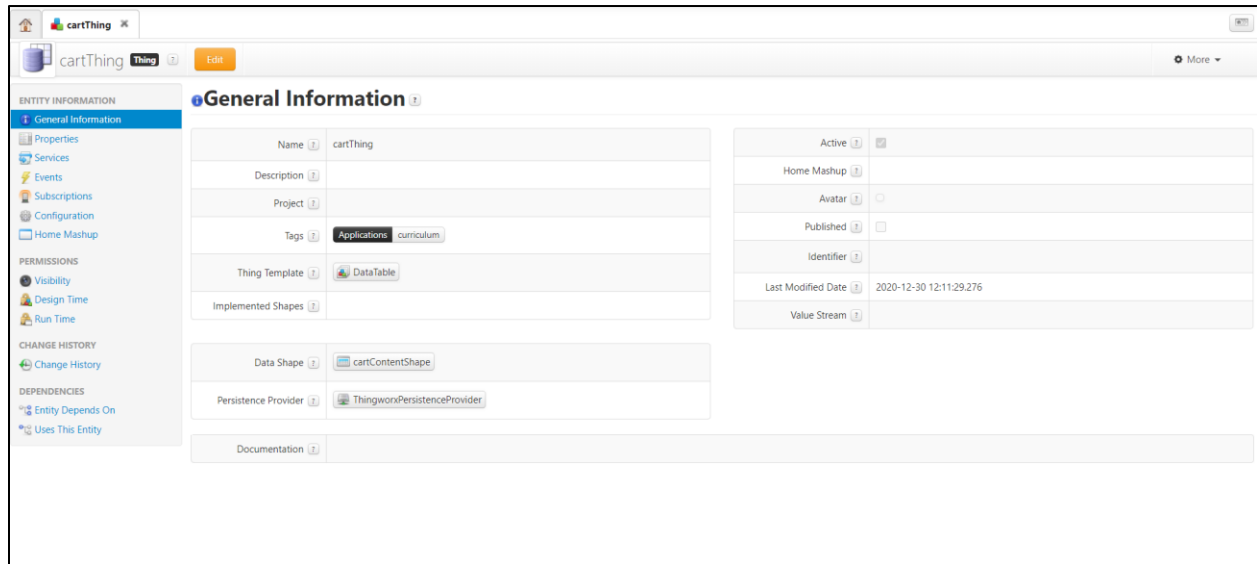
All code comments follow `this convention`

303.1 Becoming Familiar with cartThing

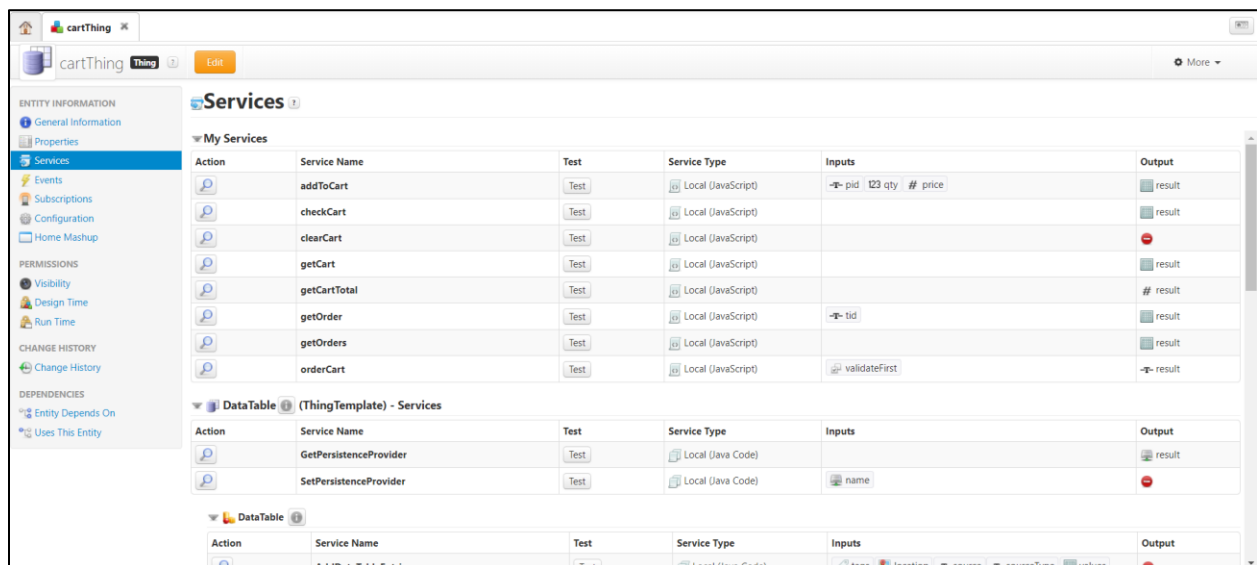
Like in Metadata 302, ThingWorx will be connected to Vuforia Studio. This time, in addition to the **shoppingThing**, a new Thing has been added, **cartThing**. **cartThing** consists of 8 services: `addToCart`, `checkCart`, `clearCart`, `getCart`, `getCartTotal`, `getOrder`, `getOrders`, and `orderCart`. Each of these services serves a different functionality, but all come together to create a persistent cart inside ThingWorx.

1. Download the **metaShoppingEntities.twx** file that has been included with this section.

- Follow the instructions for importing and exporting files into ThingWorx Composer from the [PTC Support website](#).
- Open **cartThing** once it has been added into your ThingWorx instance.
- The **General Information** tab will include general information about the Thing. in this case, the **Name**, **Description**, **Tags**, and **ThingTemplate** for the Thing are included. **Tags** are used to group or categorize ThingWorx entities and **ThingTemplates** are used to create a new Thing based on a common base and functionality.



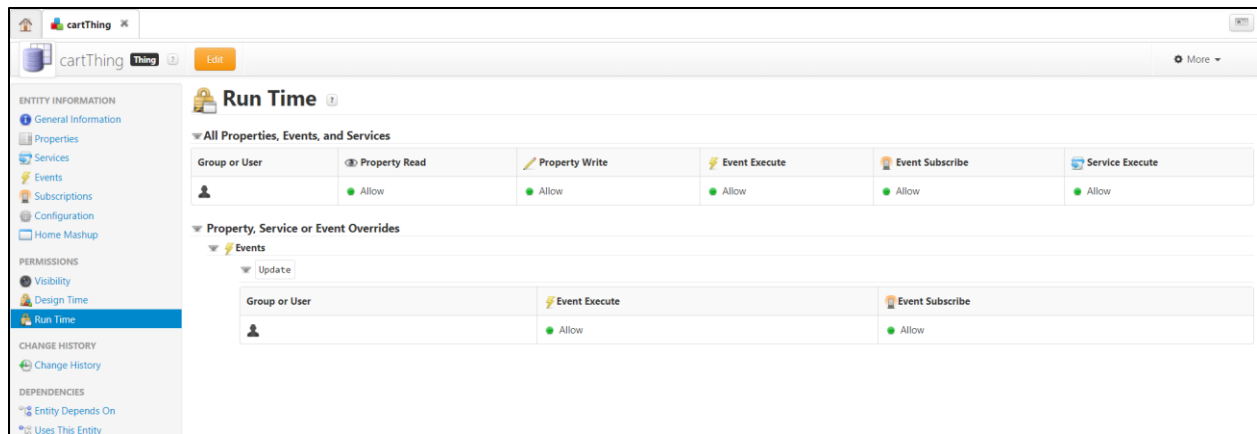
- Open the **Services** tab to view all services associated with cartThing.



- The **addToCart** service works much like the **addToCart** function that you created in Vuforia Studio. It intakes the part number (pid), quantity (qty) and price (price) of the part that has been clicked on. Using that input

information, along with user information from ThingWorx, it creates a user ID as a way of designating the person using the service. This user ID (uid) is a unique identifier for each order and is necessary for persistence because it gives the ability to tell the user and their individual sessions apart from others. When a part is added to the cart, its part information is stored inside an entry in a data table, which is then sent into the infotable used to store the cart.

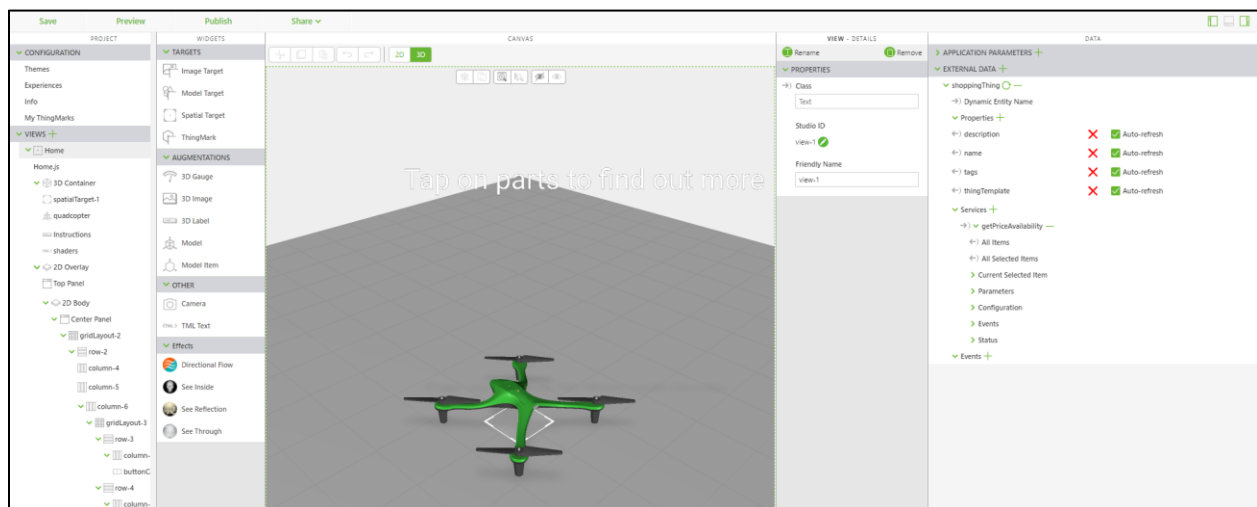
- b. Once a cart has been created, the **getCart** service is used to obtain the entries for the cart. It takes the information that was added to the cart and outputs it as an infotable. This will be used to update the cart when changes are made to it in the Vuforia Studio experience.
 - c. **getCartTotal** works like **getCart**, except instead of returning an entire table of information, its only output is the total price of the items in the cart.
 - d. When values are input into **addToCart**, there may be mistakes, like in the event of an incorrect price quote for a customer, so the **checkCart** service is used to validate all product information that was input into **addToCart**. If any information is incorrect, **checkCart** will correct that information inside the cart.
 - e. **clearCart** works exactly like its name suggests, it clears all items from the cart.
 - f. Like **clearCart**, **orderCart** is also relatively self-explanatory. When invoked, this service will place an order for all the parts that are in the cart. You are given the option of checking a box to validate the information in the cart using **checkCart**. After this service is invoked, a transaction ID (tid) is created with the user's name and number order that it is, and the **purchased** property of the infotable is changed to true to signify that the part in the cart has been bought.
 - g. **getOrder** allows you to input a transaction ID (tid) and receive all the information about that specific order in a table.
 - h. **getOrders** skips the tid input and displays all orders that have been made from the cart in a table.
6. Open the **Run Time** tab under **Permissions** and make sure the account for your ThingWorx and Vuforia Studio instances has full access permissions for this Thing. This will enable the service to be called in Vuforia Studio.



303.2 Add cartThing to Vuforia Studio

In the same manner that you added **shoppingThing** to Vuforia Studio in the previous exercise, **cartThing** will now need to be added so its services can be accessed inside Studio.

1. In your Vuforia experience, open the **Info** tab and ensure that the **Experience Service** URL is the same for your Vuforia Studio and ThingWorx instances. This is necessary for being able to connect the ThingWorx service into Studio.
2. In the **Home** tab, open the **Data** panel. Open the **External Data** dropdown if it isn't already open. You should see **shoppingThing** in the tab already.



3. Click the green + next to **External Data** to open a dialogue box for adding a new Thing.

Add External Data

Entities

actionLogging (Thing)

AlertFunctions (Resource)

AlertHistoryStream (Thing)

AlertProcessingSubsystem (Subsystem)

AnalyticsGateway (ThingTemplate)

AnalyticsJobServer (ThingTemplate)

PROPERTIES

SERVICES

EVENTS

Close

- a. Type in *cartThing* into the **Entities** search bar. This will bring up a list of properties, services, and events for the Thing.

Add External Data

Entities

cartThing

cartThing (Thing)

PROPERTIES

SERVICES

EVENTS

description

name

numberOfOrders

tags

thingTemplate

+

+

+

+

+

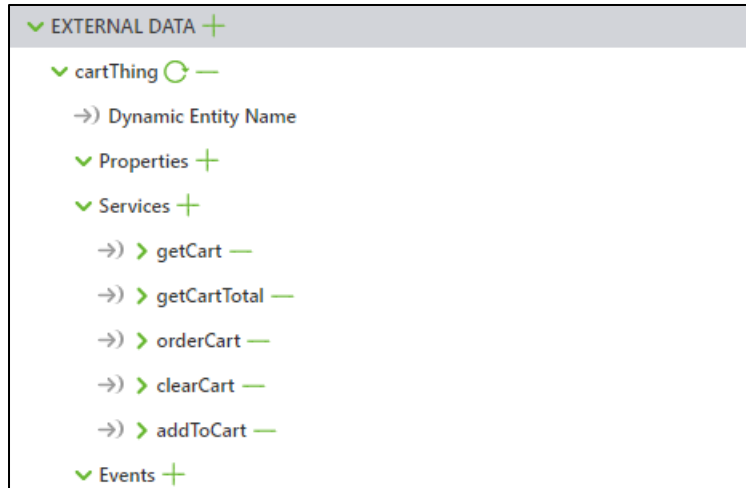
4. Click on the **Services** tab to open a list of services that can be added.

Add External Data		
Entities	PROPERTIES	SERVICES
<input type="text" value="cartThing"/>	<input type="text"/>	
cartThing (Thing)	AcknowledgeAlert	+
	AcknowledgeAllAlerts	+
	AddBooleanValueStreamEntry	+
	AddDataTableEntries	+
	AddDataTableEntry	+
	AddDateTimeValueStreamEntry	+
	AddDesignTimePermission	+
	AddDynamicSubscription	+
	AddEventDefinition	+
	AddImageValueStreamEntry	+
	AddInfoTableValueStreamEntry	+
Close		

- To find the correct services to add, type in *getCart* (both *getCart* and *getCartTotal* should appear when you do this), *orderCart*, *clearCart*, and *addToCart*. Add each service by clicking the green + next to its name. Click **Close** once you have added all the services to the experience.

Add External Data		
Entities	PROPERTIES	SERVICES
<input type="text" value="cartThing"/>	<input type="text" value="getCart"/>	
cartThing (Thing)	getCart	✓
	getCartTotal	✓

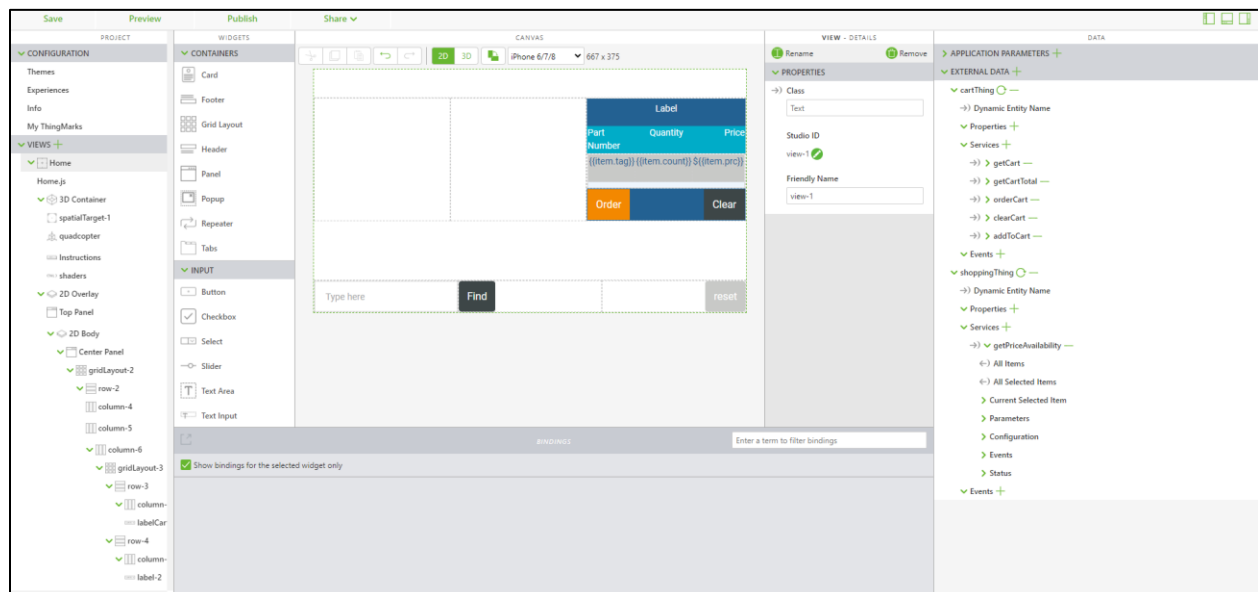
- If completed correctly, **cartThing** will look like the following in the **External Data** tab (**Services** dropdown bars have been collapsed for visibility in the image). If it has all five services associated with it, then cartThing has been added successfully.



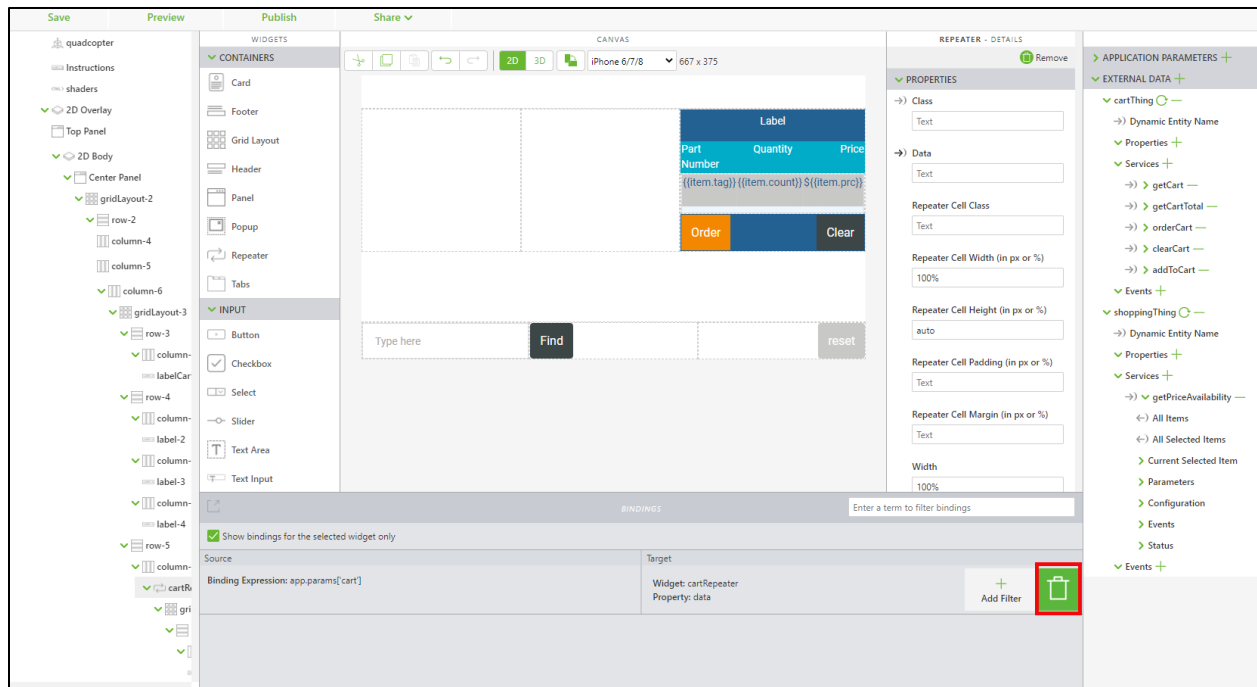
303.3 Binding cartThing and Editing the 2D Canvas

After **cartThing** has been added to the experience, its content needs to be bound to other properties and widgets for it to be effective. This section will walk you through that process, ensuring that ThingWorx and Vuforia Studio are properly connected. Some small changes will also need to be made to the 2D canvas to account for the new method of bringing in information.

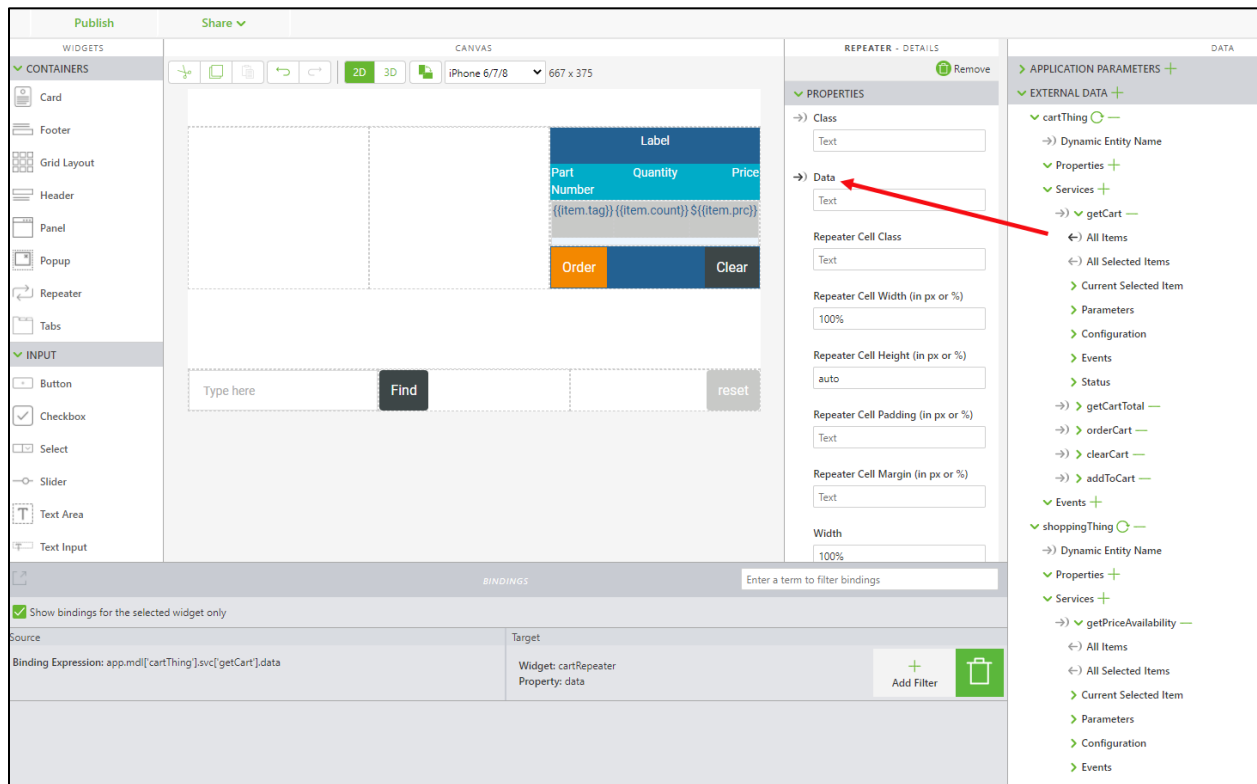
1. Open the the 2D canvas and then the **Bindings** panel. This will allow you to edit property bindings.



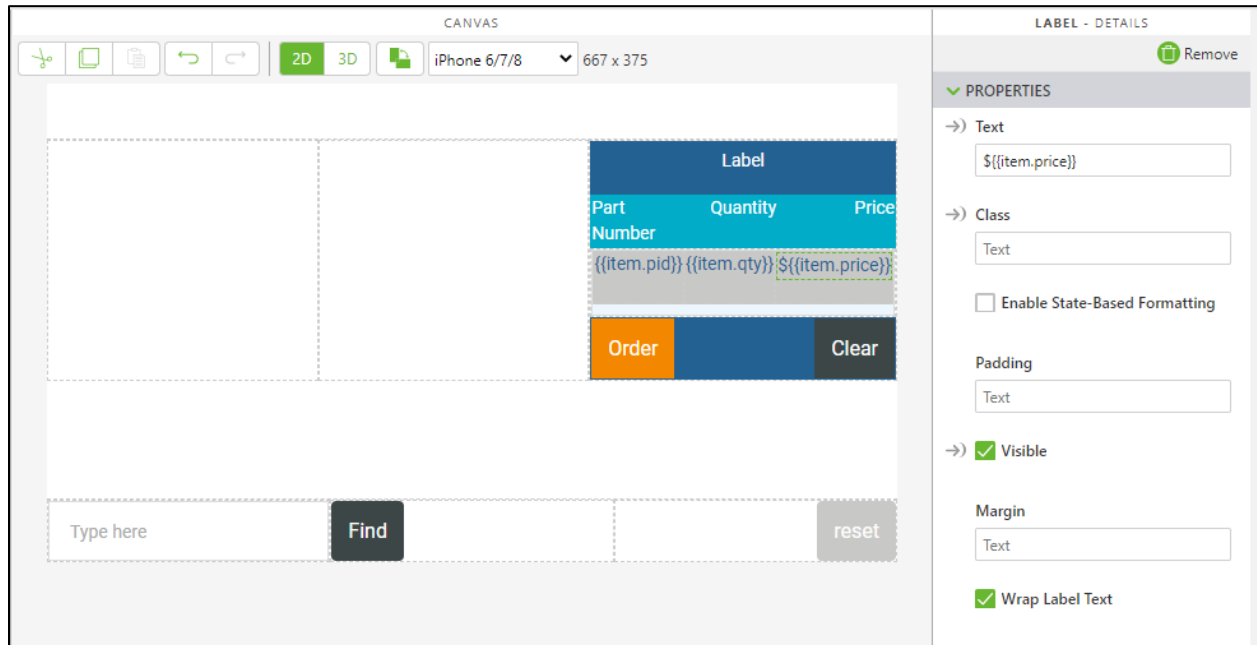
2. With the **Data** panel still open and **cartThing** visible in **External Data**, open the **cartRepeater** widget in **Home**.
 - a. From previous activities, the **Data** property of **cartRepeater** has been bound to the **cart** application parameter. Click the green trash can icon in the **Bindings** panel to delete that binding.



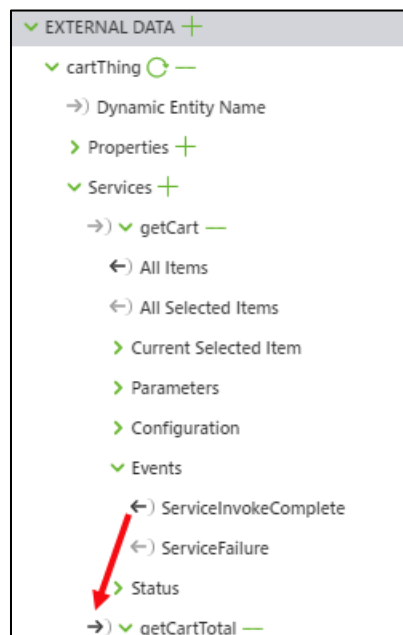
- b. Expand the dropdown for the **getCart** service. Drag **All Items** for the service onto the **Data** property of **cartRepeater** to bind them together. This binding will send data from **getCart** to the repeater so it can be displayed when the repeater is updated.



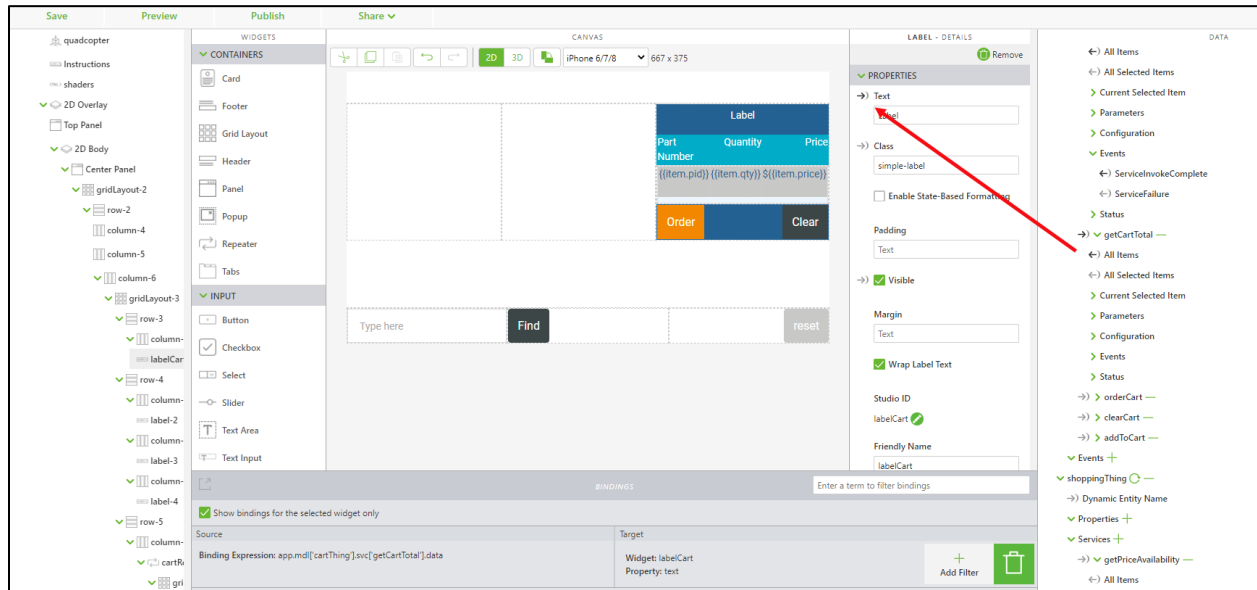
3. Change the **Text** property for the 3 labels inside the repeater from `{{item.tag}}`, `{{item.count}}`, `{{item.prc}}` to `{{item.pid}}`, `{{item.qty}}`, `{{item.price}}`, respectively. The changes will be seen later in the experience when the cart is updated.



4. Open the **Events** dropdown inside **getCart**. Bind **ServiceInvokeComplete** to the **getCartTotal** service. This binding will ensure that whenever **getCart** is completed to get the contents of the cart, **getCartTotal** will be invoked to update the total of the cart.



5. Remove the binding from the **cartLabel** application parameter from the **Text** property of the **labelCart** label widget. In its place, bind **All Items** from the **getCartTotal** service to the **Text** property. This will be used to display the total of the cart.



- a. A filter is going to need to be added so that when the total is displayed in the label, it has the same format before of Cart(\$total). A filter is used instead of typing out the text because this information is coming from a ThingWorx service. In the **Target** box for the binding between **getCartTotal** and **Text** for the label, click **Add Filter**. This will open a dialogue box for editing information about a filter.

The 'Add Data Filter' dialog box is shown. It has a title bar 'Add Data Filter'. Below the title bar, there is a 'Filter Name' label and an empty text input box. Below that is a 'Filter Body' label and a larger empty text area. At the bottom of the dialog, there are two buttons: a green 'Done' button and a green 'Cancel' button.

- b. Type *total* in the **Filter Name** box. In the **Filter Body** box, add the following line of code: `return 'Cart ($' + value[0].result + ')';`. this filter

will take the value that is output from **getCartTotal** and add it into a string to show the total of the cart. Click **Done** when you have completed this step.

Add Data Filter

Filter Name

Filter Body

```
return 'Cart ($' + value[0].result + ')';
```

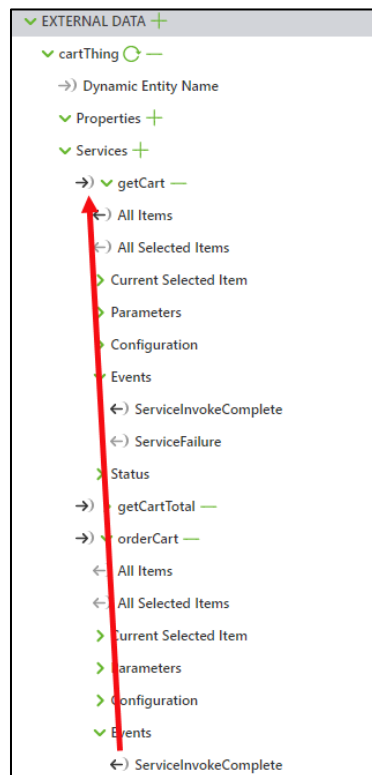
Done

[Cancel](#)

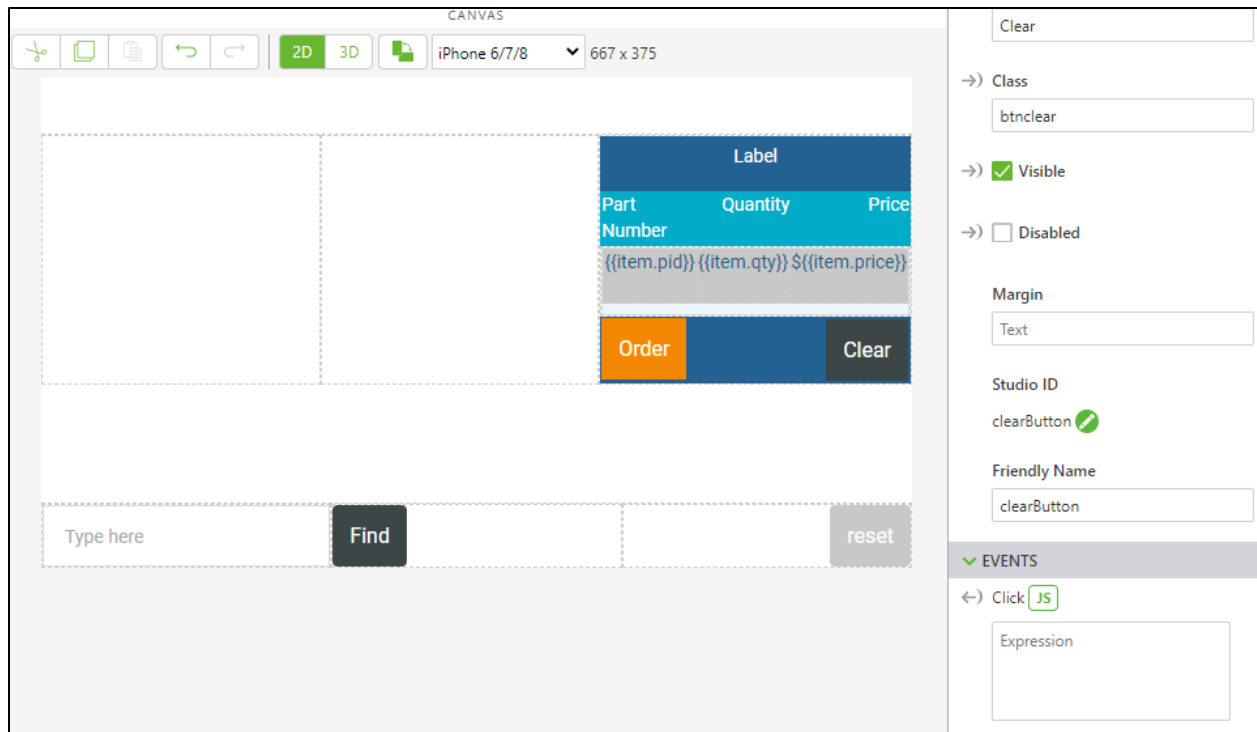
6. Open the properties for the **Order** button.
 - a. Bind the **Click** event for the button to the **orderCart** service that was added. Now, when the **Order** button is clicked on, the **orderCart** service will be invoked in ThingWorx and order the cart.

The screenshot displays the ThingWorx Studio interface. On the left, the 'WIDGETS' panel shows various containers and input widgets. The central 'CANVAS' area shows a table with columns 'Part Number', 'Quantity', and 'Price', and buttons 'Order' and 'Clear'. The right panel, 'BUTTON - DETAILS', shows the properties and events for the 'Order' button. The 'EVENTS' section shows the 'Click' event bound to the 'orderCart' service. A red arrow points from the 'Click' event to the 'orderCart' service in the 'DATA' panel on the far right. The 'BINDINGS' panel at the bottom shows the binding for the 'Order' button, with the source being 'Label: Button' and the handler being 'Service: orderCart'.

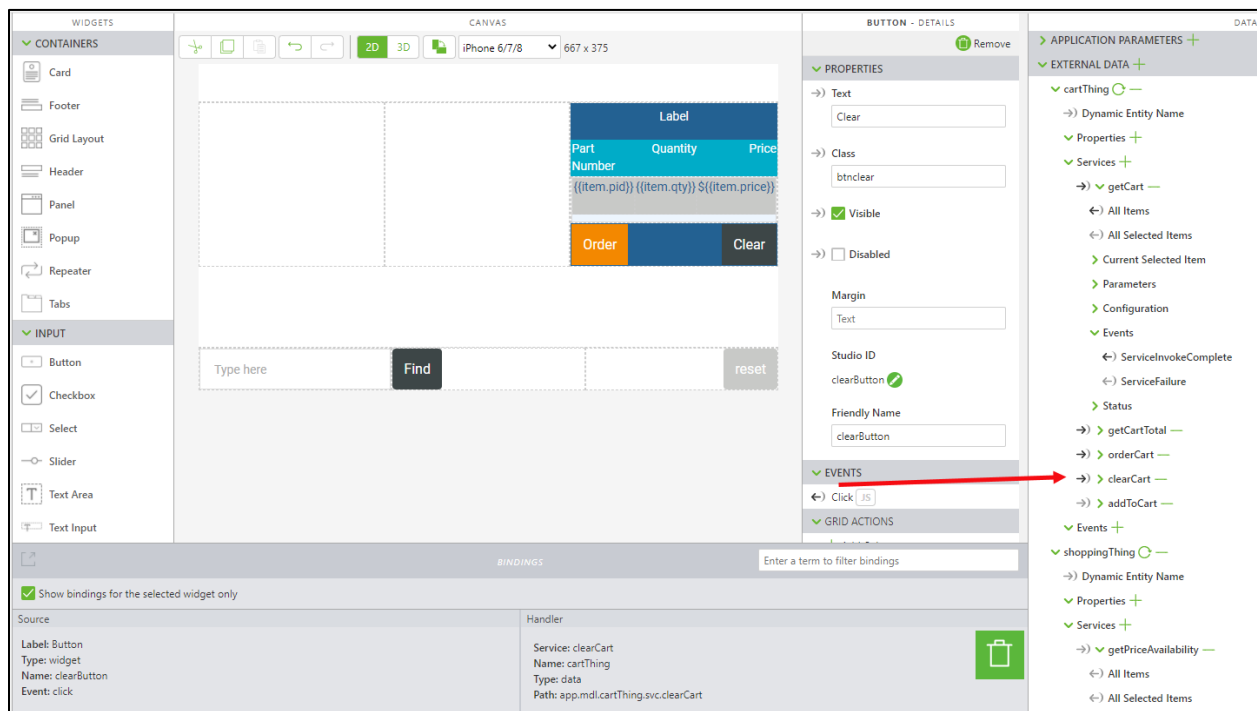
- b. Additionally, bind the **ServiceInvokeComplete** event for **orderCart** to the **getCart** service. This will cause **getCart** to activate every time the **orderCart** service is completed, updating the cart with its current contents.



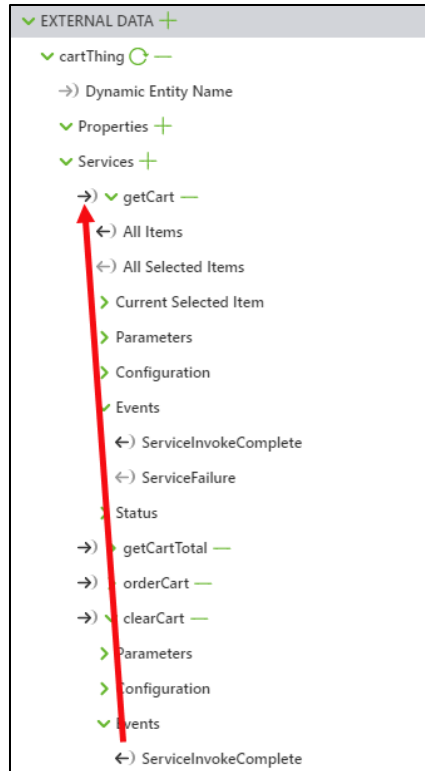
7. Like the **Order** button, the same action will be taken with the **Clear** button.
- a. Open the **Clear** button and remove the text in the **JS** section of the **Click** event since the **clearCart** service is going to be used.



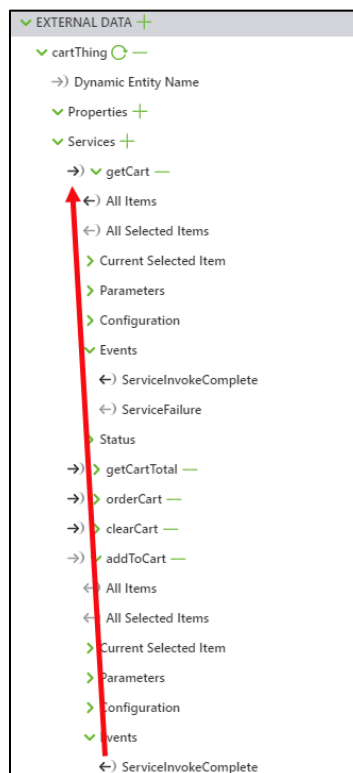
- b. Bind the **Click** event to the **clearCart** service. Clicking the **Clear** button in the experience will now clear all items from the cart.



- c. Bind the **ServiceInvokeComplete** event for **clearCart** to **getCart** as well, just like with **orderCart** in the previous step. This will also update your cart with **getCart** whenever the cart is cleared.



- As the final binding, bind the **ServiceInvokeComplete** event for **addToCart** to **getCart**. Like with **orderCart** and **clearCart**, **getCart** will be triggered to update the displayed cart in the experience whenever a new part is added to the cart.



303.4 Invoking the addToCart Service, Accounting for Welded Parts, and Cleaning Up the Code

Once the bindings have been created, the last step to creating this persistent cart is calling the **addToCart** service using the popup that appears when a part is selected. This is what will lead to the part being added to the cart inside Thingworx. This section will also make a callback to Metadata 101, when you combined the battery and PCB into the electronics part. Additionally, code that is not needed anymore will be deleted to keep the code as clean as possible.

1. Open **Home.js**.
2. In order to clean up some code, the **priceInfo** application parameter will be deleted from the code. It is not needed anymore since the total of the cart will be calculated in ThingWorx.
3. With **priceInfo** not part of the code anymore, the price variable needs to be changed to **\$scope.price** to make it accessible throughout the code. Additionally, the **'\$'** can be removed from the conditional statement, as that will be added in the text of the popup itself and in the label filter.

```
52 //  
53 // price is going to be the variable that is referenced in the popup, whi  
54 $scope.price = rowData.avail === true ? rowData.price  
55 : 'UNAVAILABLE';  
56  
57 //  
58 // create a variable to bring the $scope.meta object into this event list  
59 let meta = $scope.meta
```

- a. Inside the template for the popup, change the **price** input for **setTemplate** to be **\$scope.price** to account for its new scope.

```
//  
// call the setTemplate function to populate the template  
template: $scope.setTemplate(meta, $scope.price),
```

```
63 $scope.popup = $ionicPopup.show({  
64  
65 //  
66 //template for the popup with added buttons  
67 template: $scope.setTemplate(meta, $scope.price),  
68  
69 scope: $scope  
70 }); //end of ionic popup
```

- b. Navigate to the **setTemplate** function. In the **addTo** variable, add a **'\$'** to the variable so the popup will display the dollar sign before **price** in the case that the part is available.

```
//  
// if price != unavailable, define an add to cart button and have the price displayed  
in the popup, if it is unavailable, just display price  
var addTo = price != 'UNAVAILABLE' ? '$' + price + '&nbsp;&nbsp;&nbsp;</div><div class="btnadd" ng-  
click="hiliteOff();popup.close();addToCart();">Add to Cart</div>  
: price ;
```

```
287 //  
288 // if price != unavailable, define an add to cart button and have the price displayed in the popup, if it is unavailable, just display price  
289 var addTo = price != 'UNAVAILABLE' ? '$' + price + '&nbsp;&nbsp;&nbsp;</div><div class="btnadd" ng-click="hiliteOff();popup.close();addToCart();">Add to Cart</div>  
290 : price ;
```

4. Scroll up to your **addToCart** function.

- a. The **addToCart** service has the same functionality as the `addToCart` function, so remove the entire body of the function so it just looks like the code below.

```
//
// function for adding a selected part to the cart
$scope.addToCart = function () {

} //end of addToCart function
```

```
202 //
203 // function for adding a selected part to the cart
204 * $scope.addToCart = function () {
205
206 } // end of addToCart function
```

- b. Use the `triggerDataService` function to call the `addToCart` service from ThingWorx. Use the variables below for the `pid`, `qty`, and `price` inputs.

```
//
// call addToCart service from ThingWorx
twx.app.fn.triggerDataService('cartThing', 'addToCart', {pid: $scope.meta.partNumber,
qty: 1, price: $scope.price});
```

```
202 //
203 // function for adding a selected part to the cart
204 * $scope.addToCart = function () {
205
206 //
207 // call addToCart service from ThingWorx
208 twx.app.fn.triggerDataService('cartThing', 'addToCart', {pid: $scope.meta.partNumber, qty: 1, price: $scope.price});
209
210 } // end of addToCart function
```

5. Remove the `clearCart` function entirely, since it has been transitioned to be a service in ThingWorx.
6. Also, remove the code for creating the `cartLabel` application parameter and the `cart` object, as those are not necessary anymore.
7. In order to access the pricing data for the electronics combined part, logic needs to be implemented to bring the occurrence data up a level in its structure. This will allow for data about the electronics part to appear in the popup when you click on either the PCB or battery.
 - a. Add this code above `$scope.meta`.
 - b. The code below works in the following way; a variable named `welding` is created that triggers the `while` loop to occur. Inside the `while` loop, a variable named `sbominfo` is created, which is set with the value of the `sBOM_Welded` attribute. If `sbominfo` has information in it and the `sBOM_Welded` attribute had a value of `true`, then the index of the occurrence will be rolled back a level to the parent part of the selected parts. In other words, since electronics is considered to be the parent part of the PCB and battery, whenever the PCB or battery is selected, the occurrence path (`pathId`) will become the path of the electronics part. If the selected part does not have the `sBOM_Welded` attribute, then `welding` is set to `false` and the `while` loop ends.

```
//
// variable to pull the value for the occurrence property in the eventData JSON
object from the model. Create variable for the currently selected part
var pathId = JSON.parse(eventData).occurrence
```

```
$scope.currentSelection = targetName + "-" + pathId
```

```
//  
// set welding = true until it is turned false  
var welding = true  
while (welding) {  
  
    //  
    //if the part is welded, use its parent part  
    var sbominfo = metadata.get(pathId, 'sBOM_Welded');  
    if (sbominfo != undefined && sbominfo === 'true') {
```

```
        //try parent until the root is reached  
        var child = pathId.lastIndexOf('/');  
        if (child === 0)  
            return;
```

```
        pathId = pathId.substr(0, child);
```

```
    } else {  
        welding = false;  
    } // end of if statement
```

```
} //end of while loop
```

```
10 ▾ .then ( (metadata) => {  
11  
12     //  
13     // variable to pull the value for the occurrence proper  
14     var pathId = JSON.parse(eventData).occurrence  
15     $scope.currentSelection = targetName + "-" + pathId  
16  
17     //  
18     // set welding = true until it is turned false  
19     var welding = true  
20 ▾ while (welding) {  
21  
22         //  
23         //if the part is welded, use its parent part  
24         var sbominfo = metadata.get(pathId, 'sBOM_Welded');  
25 ▾ if (sbominfo != undefined && sbominfo === 'true') {  
26  
27             //try parent until the root is reached  
28             var child = pathId.lastIndexOf('/');  
29             if (child === 0)  
30                 return;  
31  
32             pathId = pathId.substr(0, child);  
33  
34 ▾         } else {  
35             welding = false;  
36         } // end of if statement  
37  
38     } //end of while loop  
39  
40     //  
41     // create an object with the properties below that are  
42 ▾ $scope.meta = {
```

- c. To see how this works, click **Save** and then **Preview**. Click on the end of the PCB. Notice that the popup now has the part name, display name, and price of the electronics combined part, as opposed to before when there was not a price displayed and the information for the PCB appeared in the popup.



8. The full code for this section can be found in [Appendix 1](#).

303.5 Using the Experience

Now that all the code has been cleaned up and the Vuforia experience is connected to ThingWorx, you will be able to see how the two systems work together.

1. Open **Preview** and add a few parts to your cart.



2. Open your ThingWorx instance in another tab and navigate back to the **Services** tab for **cartThing**.

Action	Service Name	Test	Service Type	Inputs	Output
	addtoCart	Test	Local (JavaScript)	pid qty # price	result
	checkCart	Test	Local (JavaScript)		result
	clearCart	Test	Local (JavaScript)		result
	getCart	Test	Local (JavaScript)		result
	getCartTotal	Test	Local (JavaScript)		result
	getOrder	Test	Local (JavaScript)	tid	result
	getOrders	Test	Local (JavaScript)		result
	orderCart	Test	Local (JavaScript)	validateFirst	result

Action	Service Name	Test	Service Type	Inputs	Output
	GetPersistenceProvider	Test	Local (Java Code)		result
	SetPersistenceProvider	Test	Local (Java Code)	name	result

Action	Service Name	Test	Service Type	Inputs	Output
	AddDataTableEntries	Test	Local (Java Code)	tags location source sourceType values	result
	AddDataTableEntry	Test	Local (Java Code)	tags location source sourceType values	id
	AddOrUpdateDataTableEntries	Test	Local (Java Code)	tags location source sourceType values	result
	AddOrUpdateDataTableEntry	Test	Local (Java Code)	tags location source sourceType values	id
	AssignDataTableEntries	Test	Local (Java Code)	tags location source sourceType values	result
	CreateValues	Test	Local (Java Code)		Values
	CreateValuesWithData	Test	Local (Java Code)	values	Values

- a. Click the **Test** button for **getCart**. This will allow you to run the service inside ThingWorx and view the current cart. In the window that appears, click **Execute Service** to make the cart appear. Compare your cart in ThingWorx to the repeater in Studio; notice that the two have the same items in them. Notice that **purchased** is false for all items, this is because they are in the cart, but they have not been purchased yet.

getCart - Test Service

Please be careful. Only execute services and queries where you understand the impacts.

Inputs:
No inputs

key	location	source	sourceType	tags	timestamp	pid	qty	price	uid	purchased	username	tid	available
jadelanoqc-base-1111Fri Jan 22 2021 00:10:28 GMT-0000 (UTC)	0.0000 : 0.0000	jadelano	User		2021-01-21 19:10:28.066	qc-base-1111	1	25	jadelanoqc-base-1111Fri Jan 22 2021 00:10:28 GMT-0000 (UTC)	false	jadelano		true
jadelanoqc-cover-1111Fri Jan 22 2021 00:10:32 GMT-0000 (UTC)	0.0000 : 0.0000	jadelano	User		2021-01-21 19:10:32.030	qc-cover-1111	1	14	jadelanoqc-cover-1111Fri Jan 22 2021 00:10:32 GMT-0000 (UTC)	false	jadelano		true
jadelanorotor_1234Fri Jan 22 2021 00:10:37 GMT-0000 (UTC)	0.0000 : 0.0000	jadelano	User		2021-01-21 19:10:37.716	rotor_1234	1	11	jadelanorotor_1234Fri Jan 22 2021 00:10:37 GMT-0000 (UTC)	false	jadelano		true

Execute Service Create DataShape from Result Close

- b. Click **Close** to exit the window.
3. Go back to the Studio preview tab and click the **Clear** button. You should see the repeater be cleared and the total of the cart go to **\$0**.
 - a. In ThingWorx, test **getCart** again. The cart should now appear as empty.

getCart - Test Service

⚠ Please be careful. Only execute services and queries where you understand the impacts.

Inputs:
No inputs

Results

key	location	source	sourceType	tags	timestamp	pid	qty	price	uid	purchased	username	tid	available
-----	----------	--------	------------	------	-----------	-----	-----	-------	-----	-----------	----------	-----	-----------

Execute Service Create DataShape from Result Close

- b. Click **Close** to exit the window.
4. In Studio, add the same items back to the cart.



5. This time click the **Order** button. You should see the cart clear once again.
6. In ThingWorx, click the **Test** button next to the **getOrders** service. Click **Execute Service** in the window that appears. You will see your order appear in the list. If you have made multiple orders, then multiple orders will be listed. Notice that the **purchased** property is now set to **true** because you purchased the item out of the cart, the there is a value in the **tid** column for the transaction with your login number and the order number.

getOrders - Test Service

⚠ Please be careful. Only execute services and queries where you understand the impacts.

Inputs:
No inputs

Results

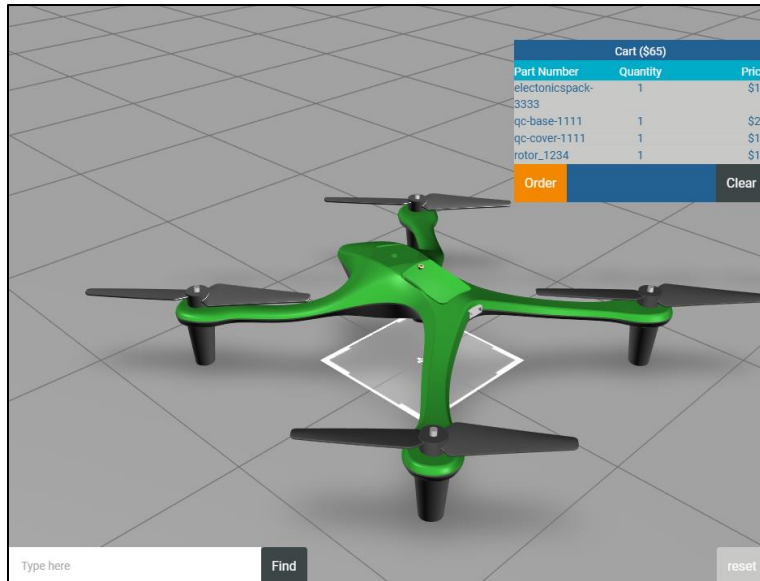
key	location	source	sourceType	tags	timestamp	pid	qty	price	uid	purchased	username	tid	available
jadelanoqc-base-1111Fri Jan 22 2021 00:10:28 GMT-0000 (UTC)	0.0000 ; 0.0000	jadelano	User		2021-01-21 19:12:25.648	qc-base-1111	1	25	jadelanoqc-base-1111Fri Jan 22 2021 00:10:28 GMT-0000 (UTC)	true	jadelano	jadelano25	true
jadelanoqc-cover-1111Fri Jan 22 2021 00:10:32 GMT-0000 (UTC)	0.0000 ; 0.0000	jadelano	User		2021-01-21 19:12:25.650	qc-cover-1111	1	14	jadelanoqc-cover-1111Fri Jan 22 2021 00:10:32 GMT-0000 (UTC)	true	jadelano	jadelano25	true
jadelanorotor_1234Fri Jan 22 2021 00:10:37 GMT-0000 (UTC)	0.0000 ; 0.0000	jadelano	User		2021-01-21 19:12:25.651	rotor_1234	1	11	jadelanorotor_1234Fri Jan 22 2021 00:10:37 GMT-0000 (UTC)	true	jadelano	jadelano25	true

Execute Service Create DataShape from Result Close

7. Continue to test out the services. In Studio, add the same parts that you added before to the cart.



- a. This time, after doing that, close out of Vuforia Studio and then open it back up again. Open the preview of the experience and add a new part. You will see that since you did not clear your cart or order the parts before closing the window that they stayed in the cart. This is exactly how persistence works.



- If the persistent cart is working, then you have completed this tutorial and our Metadata series! When trying to publish this experience, make sure the **Access** dropdown bar in the **Info** tab is set to **Requires Authentication**. This is necessary for validating that your system will connect to ThingWorx from your mobile device.

Save
Preview
Publish
Share

PROJECT

CONFIGURATION

Themes
Experiences
Info
My ThingMarks

VIEWS

Home

Home.js

3D Container

spatialTarget-1

quadcopter

Instructions

shaders

2D Overlay

Top Panel

2D Body

Center Panel

gridLayout-2

row-2

column-4

column-5

column-6

gridLayout-3

row-3

column-

labelCar

row-4

column-

label-2

Info Editor

Description
Max. 100 characters

Experience Service
Validate
☐ Allow self-signed certificates

Scan this QR code with Vuforia View (8.3.0 or later) to set this as the Experience Service URL on your device. Once this is completed, View's Library will be populated.

Access
Requires Authentication
If set to Public, users can access Experiences without authentication. Authentication may be required to view ThingWorx data depending on whether the system information, see [Configuring Public Access to ThingWorx](#) in the Help Center.
☐ Allow download for offline viewing

Minimum Screen Width
Phone (320dp)

View Navigation Menu
Enabled

Project Thumbnail
Drop image or click to upload

Appendix 1: Section 303.4 Code

```
//
// triggered when user clicks on object in the scene
$scope.$on('userpick', function (event, targetName, targetType, eventData) {

    //
    //Look at model and see if it has metadata. If it does, then execute the below code and
    create an object called metadata
    PTC.Metadata.fromId(targetName)
        .then ( (metadata) => {

        //
        // variable to pull the value for the occurrence property in the eventData JSON
        object from the model. Create variable for the currently selected part
        var pathId = JSON.parse(eventData).occurrence
        $scope.currentSelection = targetName + "-" + pathId

        //
        // set welding = true until it is turned false
        var welding = true
        while (welding) {

            //
            //if the part is welded, use its parent part
            var sbominfo = metadata.get(pathId, 'sBOM_Welded');
            if (sbominfo != undefined && sbominfo === 'true') {

                //try parent until the root is reached
                var child = pathId.lastIndexOf('/');
                if (child === 0)
                    return;

                pathId = pathId.substr(0, child);

            } else {
                welding = false;
            } // end of if statement

        } //end of while loop

        //
        // create an object with the properties below that are based on attribute names from
        Creo Illustrate for this model. Use metadata.get to obtain the data from the JSON
        properties for this occurrence.
        $scope.meta = {
            partName      : metadata.get(pathId, 'Display Name'),
            instructionName : metadata.get(pathId, 'illustration'),
            partNumber     : metadata.get(pathId, 'partNumber'),
        } // $scope.meta end

        //
        // set itemName app parameter to be equal to the partName variable, same relationship
        with itemNumber and partNumber and priceInfo and price.
        // Set the itemCount to 1 for the purpose of this section, since it is not hooked up
        to an actual inventory.
        $scope.app.params.itemName = $scope.meta.partName;
        $scope.app.params.itemNumber = $scope.meta.partNumber;
```

```

$scope.app.params.itemCount = 1;

$scope.target = targetName;

//
// call the getPriceAvailability ThingWorx service based on partNumber
twx.app.fn.triggerDataService('shoppingThing', 'getPriceAvailability', {pid:
$scope.meta.partNumber})

    }) //end brackets for PTC API and .then

//
//catch statement if the promise of having a part with metadata is not met
.catch( (err) => { console.log('metadata extraction failed with reason : ' +err) })

}) //end brackets for userpick function. Will continue to move throughout code

$scope.$on('getPriceAvailability.serviceInvokeComplete', function(evt) {

//
// variable holding all data for the current row in the infotable
var rowData = twx.app.mdl['shoppingThing'].svc['getPriceAvailability'].data.current

//
// price is going to be the variable that is referenced in the popup, while the app
parameter priceInfo will be used for adding the total in the cart
    $scope.price = rowData.avail === true ? rowData.price
        : 'UNAVAILABLE';

//
// create a variable to bring the $scope.meta object into this event listener as a
local object
    let meta = $scope.meta

//
// adds an ionic popup when a part is clicked. Show the quantity, part number, name,
and price of the selected object. &nbsp;<br> adds a line break between the two variables
    $scope.popup = $ionicPopup.show({

//
// call the setTemplate function to populate the template
template: $scope.setTemplate(meta, $scope.price),

scope: $scope
    }); //end of ionic popup

//
//highlight the chosen item and set the shader to true
$scope.hilite([$scope.currentSelection], true);

//
//function for removing the highlight
$scope.hiliteOff = function() {

    $scope.hilite([$scope.currentSelection], false)

}; // end of hiliteOff function

```

```

//
// function to be bound to the Disassemble button in the popup
$scope.disassemble = function () {

    //
    // set an object that targets the model and its instruction property
    var modelObject = {      model: $scope.targetName,
                             instruction: '1-Creo 3D - ' + meta.instructionName + '.pvi' };

    //
    // set the sequence for the quadcopter to be the name of the associated instruction
    $scope.view.wdg.quadcopter.sequence = modelObject.instruction

} //disassemble function end

}) // getPriceAvailability end

//
//function for using the userInput text box to search for parts
$scope.findMeta = function () {

    //
    //set a variable for comparing the user input to the value of the partno application
    parameter
    var searchNum = $scope.app.params.partno;

    //
    // instead of using metadata from just the picked part, use metadata from the whole
    model. If resolved, proceed
    PTC.Metadata.fromId('quadcopter')
        .then((metadata) => {

            //
            // set a variable named options. this variable will become an array of ID paths that
            fit the input text.
            // 'like' will look for a partial text match to what is typed in. use 'same' to get
            an exact match
            var options = metadata.find('partNumber').like(searchNum).getSelected();

            //
            // if the text input leads to a part number so that there is an entry in the options
            array
            if (options != undefined && options.length > 0) {

                //
                // set an empty array called identifiers. This array will house the parts that
                contain the entered part number
                var identifiers = []

                //
                // for each entry in the options array, push that value with 'quadcopter-' at the
                beginning into the ID array
                options.forEach(function (i) {
                    identifiers.push('quadcopter-' + i)
                }) //end forEach

                //
                // highlight each object in the identifiers array with the shader

```

```

    $scope.hilite(identifiers, true)

    //
    // function for removing the highlight
    var removeHilite = function (refitems) {

        //
        // return the hilite function with a value of false to the given part(s)
        return function () {
            $scope.hilite(refitems, false)
        } // end of return function

    } // end of turning off hilite

    //
    // remove the highlight of the selected part(s) after 3000 ms
    $timeout(removeHilite(identifiers), 3000)

} //end if statement

}) // end .then

//catch statement if the promise of having a part with metadata is not met
.catch((err) => { console.log('metadata extraction failed with reason : ' + err) })

} // end findMeta function

//
//sequenceloaded event listener triggers when the sequence property is updated
$scope.$on('sequenceloaded', function(event) {

    //
    // call a widget service to trigger the quadcopter model to play all steps for the
    given sequence
    twx.app.fn.triggerWidgetService('quadcopter', 'playAll');

}); //serviceloaded event function end

//
//resetit function
$scope.resetit = function () {

    //
    //set the sequence property of the quadcopter model to blank
    $scope.view.wdg.quadcopter.sequence = ''

} //resetit function end

//
// highlighting function. Inputs are the selected part and a boolean for hilite
$scope.hilite = function (items, hilite) {

    //
    //iterate over each item that is used as an imported variable for the function using
    .forEach to look at each value that comes in the items input
    items.forEach(function(item) {

        //

```

```

    //set the properties of the TML 3D Renderer to highlight the selected item using a
    TML Text shader. "green" is the name of the script for the TML Text.
    tml3dRenderer.setProperties(item, hilite === true ? { shader: "green", hidden: false,
    opacity: 0.9, phantom: false, decal: true }
                                : { shader: "Default", hidden:
    false, opacity: 1.0, phantom: false, decal: false });

    }) //foreach end

} //hilite function end

//
// function for adding a selected part to the cart
$scope.addToCart = function () {

    //
    // call addToCart service from ThingWorx
    twx.app.fn.triggerDataService('cartThing', 'addToCart', {pid: $scope.meta.partNumber,
    qty: 1, price: $scope.price});

} // end of addToCart function

//
// function for setting the template for the Ionic popup
$scope.setTemplate = function (meta, price) {

    //
    // if there is a disassembly sequence associated with the part, create a Disassemble
    button in the popup, if not, no button will appear
    var instr = meta.instructionName.length > 0 ? '<div class="btndisassemble" ng-
    click="hiliteOff();popup.close();disassemble();">Disassemble</div>'
                                : '';

    //
    // if price != unavailable, define an add to cart button and have the price displayed
    in the popup, if it is unavailable, just display price
    var addTo = price != 'UNAVAILABLE' ? '$' + price + '&nbsp;</div><div class="btnadd" ng-
    click="hiliteOff();popup.close();addToCart();">Add to Cart</div>'
                                : price ;

    //
    // build the template for the popup
    var template = '<div>' + $scope.app.params.itemCount + 'x &nbsp;' + meta.partNumber +
                                '&nbsp;</br>' + meta.partName +
                                '&nbsp;</br>' + addTo + instr +
                                '<div class="btncontinue" ng-
    click="hiliteOff();popup.close();">Continue</div>' ;

    //
    // return the template variable when this function is called
    return template

} // setTemplate end

```