



vuforia[™] studio

Metadata 202

Using JavaScript to Find Parts

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

Intro

It's common that when a part on an object breaks, new parts need to be ordered. Luckily, parts have distinct part numbers. Using a search box in this AR experience, you can enter those part numbers into a Vuforia View experience to find out the physical location of these replacement parts on the object so that they can be replaced in a quicker manner than they might if they had to be replaced with the assistance of a manual.

This portion of the project will help you become familiar with the added functionality that JavaScript coding can bring to a Vuforia Studio experience with regards to finding parts based on text input that matches attribute data and highlights the corresponding parts.

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

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

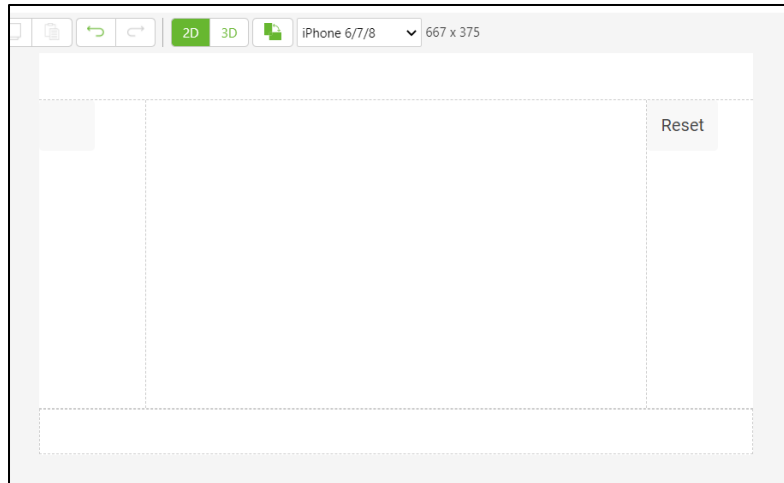
All code follows `this convention`.

All code comments follow `this convention`.

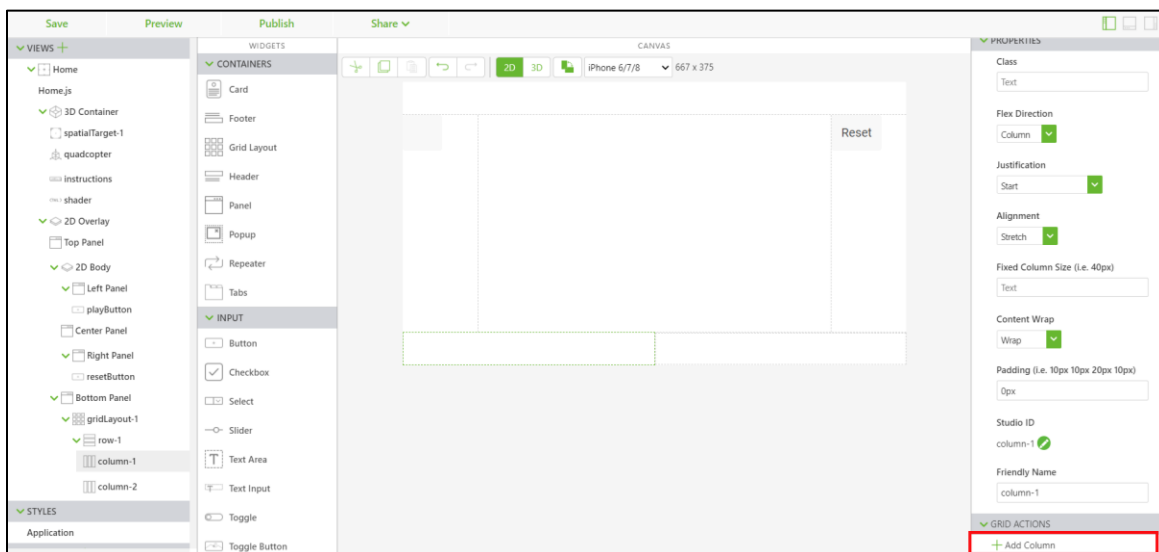
Finding Parts

In addition to being able to click on a part to view its metadata, a search bar can be added to an experience. If you had a part number but didn't know which exact part you were looking for on a model, you can search the part number and the part will be highlighted. This will be accomplished by creating a function called `findMeta` that will allow the input of a part number, and then compare it to the model data for the quadcopter. This will in turn highlight the part(s) that have the given part number.

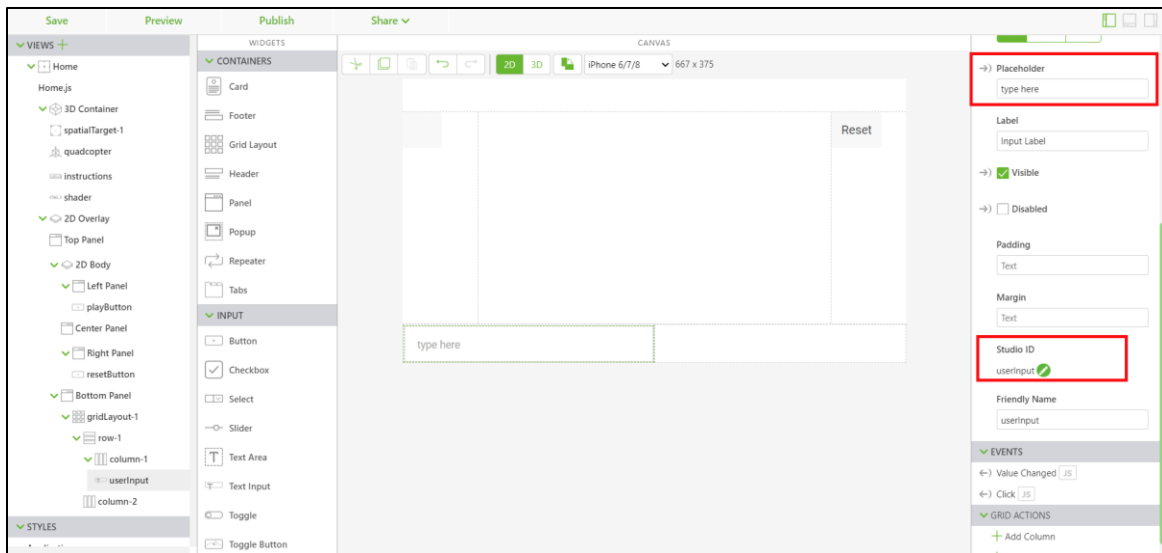
1. From the **Home** view, open the **2D** canvas view



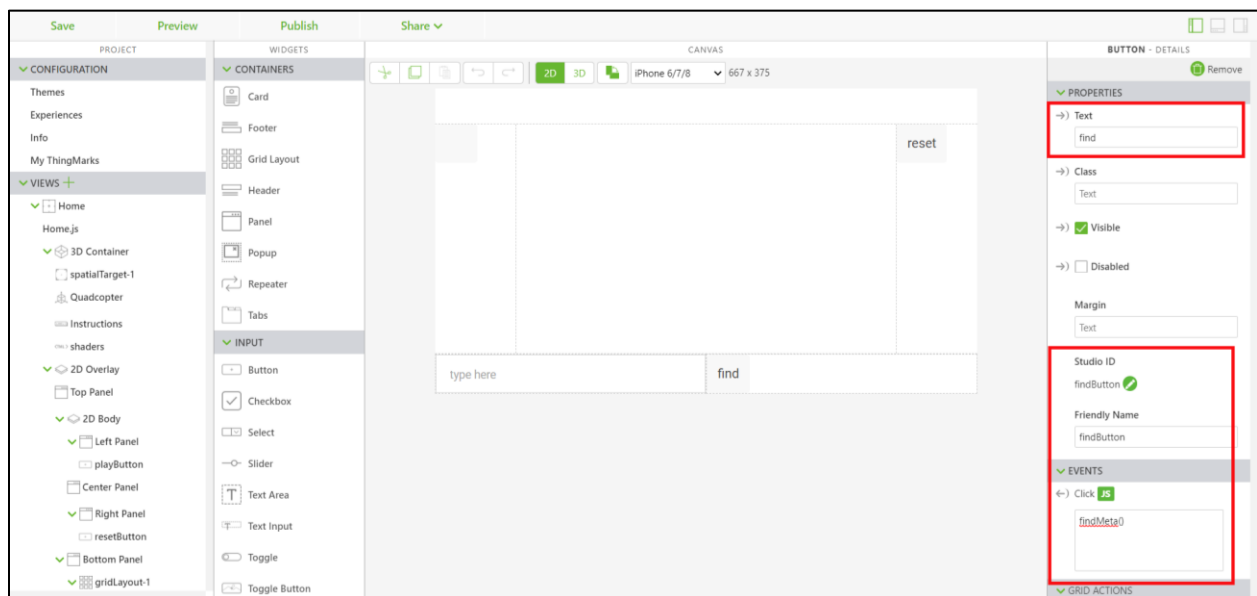
2. Drag a **Grid Layout** widget onto the bottom panel of the canvas. Click on **column-1** in the **View** tree on the left-hand side of the screen and select **Add Column** in the **Grid Actions** section to split the bottom panel into two columns.




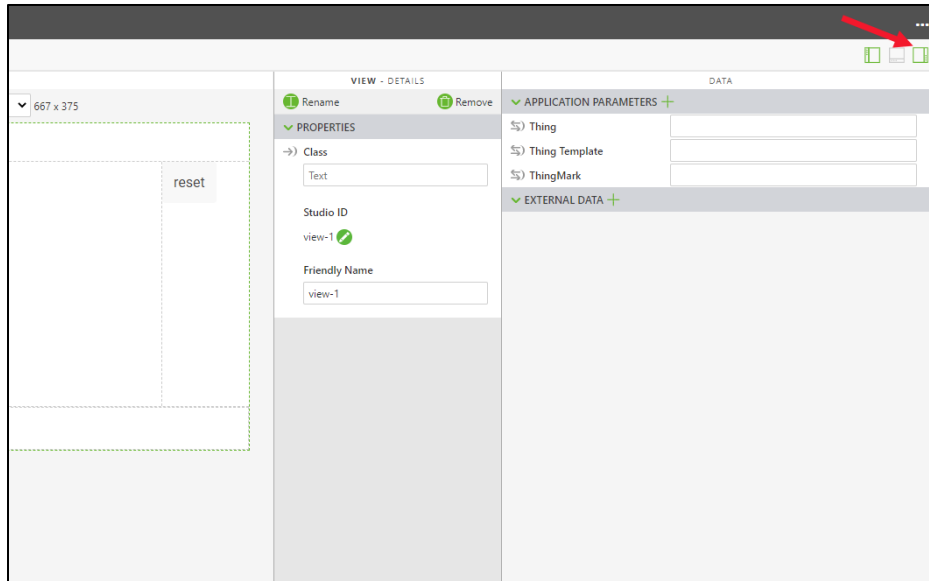
3. Drag a **Text Input** widget into **column-1** of the bottom panel. The text input will be used to enter text to search for part names or numbers. Enter *Type here* in the **Placeholder** field. Set the **Studio ID** to *userInput*.



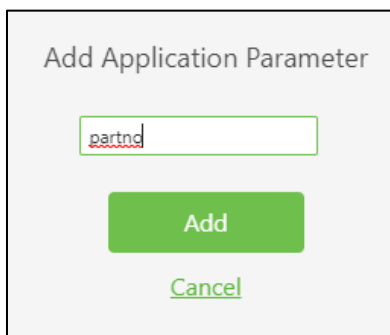
4. In **column-2**, add a **Button** widget. Enter *Find* in the **Text** field. Change the **Studio ID** to *findButton*. In the **JS** section of the **Click** event, type *findMeta()*. This function will be created in the **Home.js** tab.



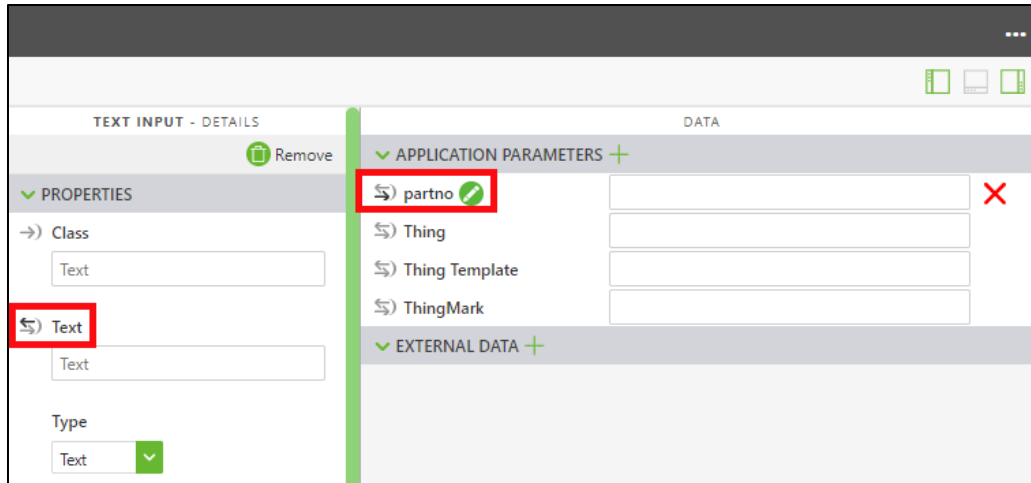
5. An application parameter needs to be created to connect the text that is typed in to the **userInput** widget to the model attributes. Click  in the upper-right corner (as shown below) to open the **Data** panel.



- a. Select the green **+** next to **Application Parameters** to create a new application parameter. Name the application parameter *partno* and click **Add**.



- b. Open the **userInput** Text Input widget. Drag and drop the **Text** property of the **userInput** widget onto the **partno** application parameter. This binds the text that is entered into the **userInput** box with the **partno** variable. The binding has been successfully created when arrows next to the two bounded objects are filled in black.



6. Click **Home.js** in the **View** tree. A new function must be created for using the search bar to find parts with a given part number. This function will take the text that is typed into the **userInput** box and set it to a variable named **searchNum**. The value of this variable will then be compared to all available part numbers in the quadcopter model. If there is a part with a part number that matches the input text, then that part, or parts if there is more than one instance of the same part number, will be highlighted using the shader from the previous Metadata 201 section. The part is highlighted for 3 seconds. Place this function after the end of the **userpick** function and before the **playit** function.
 - a. Create a function named **findMeta** that will be used to find metadata in parts that contain information that is typed into the **userInput** text box. The first step in this function is to remove the text from the play button and disassociate the model from any sequence. Next, a variable must be created called **searchNum** to have a value equal to whatever text is typed into the **userInput** text box based on the **partno** application parameter that was created.

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

  //
  //reset the text property of the play button to be blank
  $scope.view.wdg.playButton.text='';

  //
  //set the toPlay object for the play button to be undefined
  $scope.view.wdg.playButton.toPlay = undefined;

  //
  //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) => {
```

- b. The next section of the function takes the data that has been input into **userInput** and compares it to the **Part Number** attribute of the model. The **options** variable is created as an array of occurrence paths that contain data that corresponds to the text that is entered. This is done by using the **.find** and **.like** methods in conjunction with one another. When a user types text into the **userInput** box, it gets entered into the **partno** application parameter because of the binding between the text input and the parameter. Because of this, the variable **searchNum** gets set to the value of the **partno** application parameter. The **searchNum** variable is then compared to any existing **partNumber** values that are found using the **.find** method for the attributes of the model using the **.like** method. The **.like** method finds all part numbers that are either partial or exact matches to the text that is typed into the input box. These results are then stored as a list of values in the **options** variable because of **getSelected**.

```
//
// 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

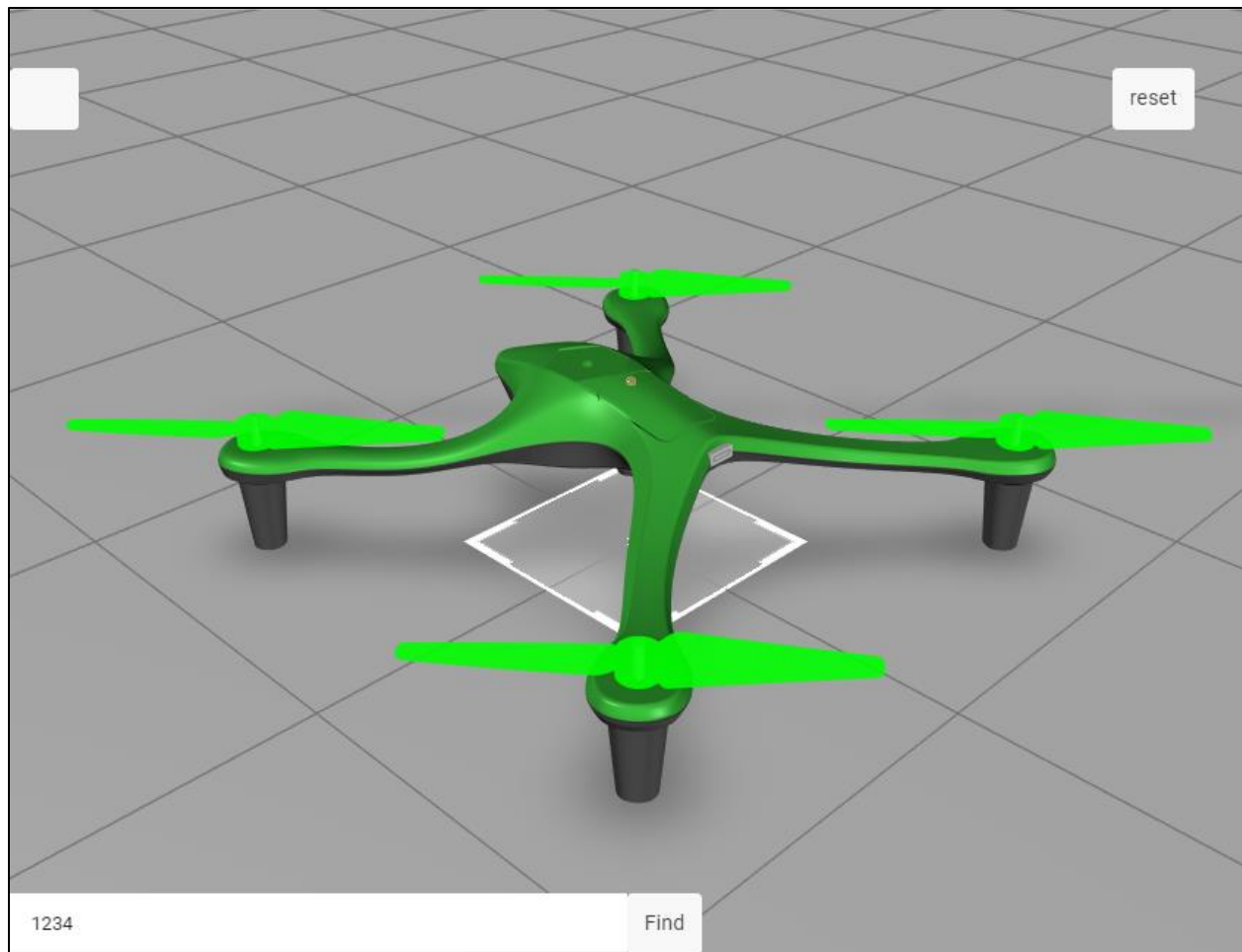
```

```

77 //function for using the userInput text box to search for parts
78 $scope.findMeta = function () {
79
80     //
81     //reset the text property of the play button to be blank
82     $scope.view.wdg.playButton.text='';
83
84     //
85     //set the toPlay object for the play button to be undefined
86     $scope.view.wdg.playButton.toPlay = undefined;
87
88     //
89     //set a variable for comparing the user input to the value of the partno application parameter
90     var searchNum = $scope.app.params.partno;
91
92     //
93     // instead of using metadata from just the picked part, use metadata from the whole model. If resolved, proceed
94     PTC.Metadata.fromId('quadcopter')
95     .then((metadata) => {
96
97         //
98         // set a variable named options. this variable will become an array of ID paths that fit the input text.
99         // 'like' will look for a partial text match to what is typed in. use 'same' to get an exact match
100        var options = metadata.find('partNumber').like(searchNum).getSelected();
101
102        //
103        // if the text input leads to a part number so that there is an entry in the options array
104        if (options != undefined && options.length > 0) {
105
106            //
107            // set an empty array called identifiers. This array will house the parts that contain the entered part number
108            var identifiers = []
109
110            //
111            // for each entry in the options array, push that value with 'quadcopter-' at the beginning into the ID array
112            options.forEach(function (i) {
113                identifiers.push('quadcopter-' + i)
114            }) //end forEach
115
116            //
117            // highlight each object in the identifiers array with the shader
118            $scope.hilite(identifiers, true)
119
120            //
121            // function for removing the highlight
122            var removeHilite = function (refitems) {
123
124                //
125                // return the hilite function with a value of false to the given part(s)
126                return function () {
127                    $scope.hilite(refitems, false)
128                } // end of return function
129
130            } // end of turning off hilite
131
132            //
133            // remove the highlight of the selected part(s) after 3000 ms
134            $timeout(removeHilite(identifiers), 3000)
135
136        } //end if statement
137
138    }) // end .then
139
140    //catch statement if the promise of having a part with metadata is not met
141    .catch((err) => { console.log('metadata extraction failed with reason : ' + err) })
142
143    } // end findMeta function
144
145    //
146    //create the playit function to bind a sequence for the model to the play button

```

7. Click **Preview**. In the **userInput** box, enter **1234** and click **find**. If all the rotors become highlighted in green and then disappear, this step has been successfully completed. Visit [Appendix 1](#) for the complete code for this tutorial. The Metadata202 folder for the completed Studio experience is also provided in GitHub.



Appendix 1: Section 5 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

            //
            // create variables based on attribute names from Creo Illustrate for this model. use
            // metadata.get to obtain the data from the JSON properties for this occurrence.
            var partName      = metadata.get(pathId, 'Display Name');
            var instructionName = metadata.get(pathId, 'illustration');
            var partNumber     = metadata.get(pathId, 'partNumber');
```

```

//
// adds an ionic popup when a part is clicked. Show the part number and name of the
selected object. &nbsp;</br> adds a line break between the two variables
var popup = $ionicPopup.show({
  template: '<div>' + partNumber + '&nbsp;</br>' + partName + '</div>',
  scope: $scope
}); //end of ionic popup

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

//
// create a function to close the popup and turn off shading. popup is the popup,
refitems is the input for the part(s) that is being highlighted
var closePopup = function (popup, refitems) {

  //
  //The function returns a method for removing the popup from the screen and turns
off the shader
  return function () {

    //
    //using the input parts, set the hilite function to be false, removing the
shading
    $scope.hilite(refitems, false)

    //
    //apply the .close method, which removes a certain section of a selected object,
to the popup variable
    popup.close()

    //
    //change the Text property of the playButton to the instructionName variable,
which was created from the JSON data of the model
    $scope.view.wdg.playButton.text = instructionName;

    //
    // create an object for the playButton called toPlay. This object will have
properties of model, which will be the name of the object that
//is clicked on and instruction, which will add the proper syntax for calling a
sequence, based off the instructionName variable, into Studio
    $scope.view.wdg.playButton.toPlay = {      model: targetName,
                                              instruction: '1-Creo 3D - ' +
instructionName + '.pvi' };

  } //return end

} // closepopup function end

//
//call the $timeout service which will call the function for closing the popup after
3 seconds (3000 ms)
$timeout(closePopup(popup, [$scope.currentSelection]), 3000);

}) //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

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

        //
        //reset the text property of the play button to be blank
        $scope.view.wdg.playButton.text='';

        //
        //set the toPlay object for the play button to be undefined
        $scope.view.wdg.playButton.toPlay = undefined;

        //
        //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

//
//create the playit function to bind a sequence for the model to the play button
$scope.playit = function () {

    //
    // if there is information in the created toPlay object to say that there is an
    illustration attribute for the part
    if ($scope.view.wdg.playButton.toPlay != undefined)

        //
        // set the sequence property for the quadcopter model to be equal to the value of the
        instruction property of the toPlay object
        $scope.view.wdg.quadcopter.sequence = $scope.view.wdg.playButton.toPlay.instruction;

    } // playit function end

    //
    //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

```