ptc

vuforia™ studio

Scaling Digital Twin Experiences 402 Extending Configurations within the Experience Server

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

Introduction

In our previous example (401), we took our two configurations of our quadcopter, and we placed the 3d model content in the Experience Server Content Deliver Service (CDS).

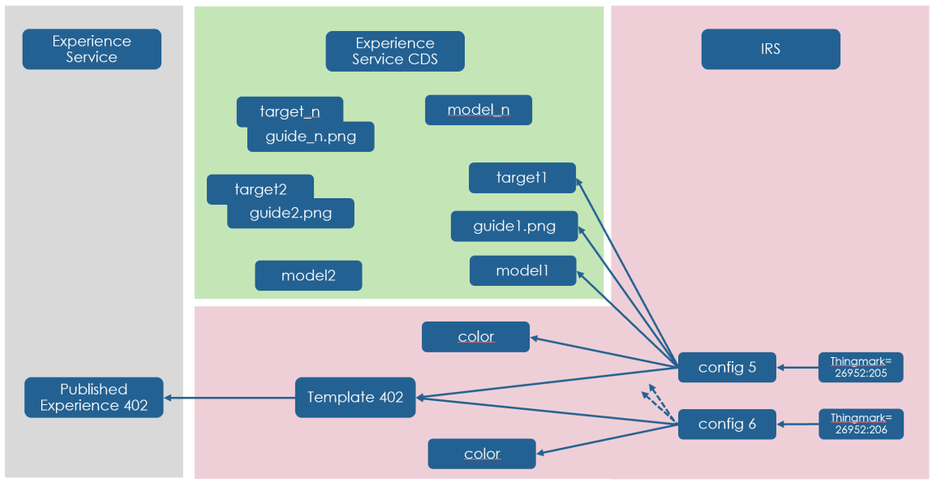
In this example, we will jump to a consumer experience and look into how you can leverage the CDS for static content, Thingworx for dynamic content, and the IRS for helping to manage configurations i.e. which combination of assets go together, and bring these into an experience which actually has very little idea of what it is viewing – it could be a quadcopter or a bicycle – it simply receives a description of the configured item and it creates the appropriate AR view.

Firstly, to help the user identify the different models, we are going to use an Image Target. Let's imagine our model quadcopter is delivered in a nice presentation box, and the box has a photo of the model on the front. A prospective customer, wishing to see what the model looks like, can scan the box and will see an Augmented 3d model appear.

Our cover photos will be used to create Image targets which are used to attach the 3D augmented model to the presentation box – the 3d model will appear to be sitting on top of the box.

In this example. We're going to use a feature of Vuforia Engine called "Image targets", which allow an image/photo to be defined as the target. Of this physical model. Or

Our models, the trackable image targets, don't change too often once the product is complete. These are perfect for locating in the Experience Service CDS. Other variables, for example color, could be stored in Thingworx as a property of a Thing that represents the Digital Twin of this model. Alternatively, we can create a key/value in the IRS that defines the color e.g. Yellow, Red etc. We can then link these to the config definition to create our final digital twin.



402.1 Creating the content package

Following on from the previous example, we will start first be preparing our content.

1. Make a new directory called “**sdte402**”
2. Inside here create the following folders
    a. “models”
    b. “images”
    c. “targets”
    d. “WEB-INF”

3. The “models” and “WEB-INF” folders are identical to the previous example – you can copy the files into this new folder structure.

```
+ models
      QuadcopterDT1.metadata.json
      QuadcopterDT1.pvz
      QuadcopterDT2.metadata.json
      QuadcopterDT2.pvz
+ WEB-INF
      metadata.json
```

4. Edit the metadata.json file, and amend the version number and description e.g.

```
{
  "version": "1.0.1",
  "title": {
    "en": "Quadcopters"
  },
  "description": {
    "en": "Models and targets of various quadcopters for exercise 400"
  }
}
```

5. In the “images” folder, we will place the images that form our product box (see example above). To create these images, take a look back at example 201, where we used Creo illustrate to create our 3D model of the quadcopter.
    a. Load **quadcopterDT1.c3di** into Creo Illustrate, and create a new Figure.
    b. New Figure opens up a dialog asking you to select a starting view – choose the **3D Default** option**.** Click **“Create and Close”**
    c. By default, this is given name “Figure 1”, so lets rename this “box-cover”. You can right-click on the figure name an choose **Rename** or you can click the Rename option in the Figure menu.
    d. Using the mouse in the 3D window, position / orient the quadcopter model on the screen – find a nice angle that makes it look nice”
    e. At the bottom of the 3D View, there will be a notice saying “No Figure View has been set”. Once you’ve found your perfect view, click the **Use the current View** to record this. The Figure will now remember this view.
    f. Finally, lets export this image to a file – we’re going to use this to define an image target. With our “box-cover” figure open in the viewer, click **File > Save Figure As > Save as Image File.** Choose the ‘images’ folder we’ve just created, and save this file as `imgDT1.png`**.**

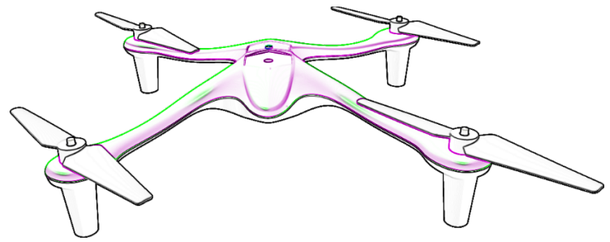
Note : example images (and the targets subsequently generated from these) are included in the examples that you have downloaded. These instructions are included here to help the reader understand the steps taken to generate these – you can skip steps 5,6,7,8 if you want to get the example working first – come back later, perhaps with you own model, and use these steps to customize the example for your own usecase.

6. Repeat the same steps for quadcopterDT2, saving the resulting file as `imgDT2`. Our images folder should look like this:-

```
+ images
        imgDT1.png
        imgDT2.png
```

7. In the “targets” folder, we will place the image target that we will create using the Vuforia image target generator. Along with the target itself, we have a guide view that will be used to help the user identify where to point their device. We will create the guide view from the same image that we use to create the target.

   guideDT1.png is a copy of imgDT1, saved from illustrate in the previous step. We use a different file here because we might want to treat the guide differently e.g. we might want an outline of the image, or perhaps a cartoon shaded image. It’s up to you what your guides look like – in the examples provided, we’ve used an outline approach. This was created using a standard image editing tool, for example Adobe Photoshop or equivalent.



   To associate a guide view to a target, we start by creating two folders under the targets folder – let’s name these quadDT1 and quadDT2. Place the two guide images into these folders.

```
+ targets
   + quadDT1
        guideDT1.png
   + quadDT2
        guideDT2.png
```

Note : this step is optional – working targets are provided in the folders provided. The information here is provided for cases where the reader may wish to learn how to create their own targets.

Image targets can be created using the Vuforia Target generator, available in the Vuforia Developer Portal. You can follow the detailed instructions here :- https://library.vuforia.com/getting-started/vuforia-target-manager

Lets start with imgDT1, which we created in step 5f. This will be our ‘target’ image.

a. Log in into the Vuforia Developer Portal
b. Navigate to the Target Manager page – here are listed all your Vuforia targets. We’re going to create 2 new targets, one per image. Lets start with imgDT1.
c. Click ‘Add Database’ to create a new target database ; name this ‘quadDT1’. Select ‘device’ as the type, and click ‘create’. A new database will be added to your list of target databases.
d. Click on the named link (quadDT1) to navigate into this target database. Click ‘Add Target’ and browse to select imgDT1 in the folder structure we defined in 5f. Select type=’single image’ and name this entry imgDT1.
e. With image targets, it is important to define the physical size that the image will appear at – this will ensure your augmentations are displayed at the correct size. For now, lets assume we’ll print the image onto a box that is US-Letter sized 8.5”x11” – we need to enter the width (11” = 28cm) in meters, so in the width box, enter 0.28.
f. Click ‘Add’ to add the target. Your target image will be uploaded, analysed and an image target will be created. The ‘Rating’ factor provides valuable feedback as to the quality of this image i.e. how recognizable and ‘trackable’ the image will be.

g. The final step is to download the database – click ‘Download databases (all)’, use the (default0 setting of ‘Android Studio, XCode or Visual Studio’ and click ‘Download’. A zip file containing the target database will be downloaded – typically to your ‘Downloads’ folder on your computer.
h. Take this zip file and move it (cut/paste, drag drop etc.) from the Downloads folder to the ‘targets’ folder that we created in step 7.
i. If you are on Windows, you can right-mouse click on the zip file and select ‘Extract all’ and choose a location for the contents. Choose a subfolder with the name of the zip file, e.g. quadDT1 – the location where we copied our guide view (step 6). After extraction, the folder

```
+ targets
      + quadDT1
            guideDT1.png
            quadDT1.dat
            quadDT1.xml
```

j. Repeat the same steps for imgDT2 (creating quadDT2)

```
+ targets
      + quadDT1
            guideDT1.png
            quadDT1.dat
            quadDT1.xml
      + quadDT2
            guideDT2.png
            quadDT2.dat
            quadDT2.xml
```

8. Your folder structure should look like this :-

```
+ images
+ models
+ targets
+ WEB-INF
```

9. As described in example 401, create a zip file that contains the 4 directories and contents. The zipfile should retain the same name as before e.g. **sdte400.zip**

402.2 Uploading content to the CDS

1. In example 401, we created a representation and named it **sdte400**. We now need to ***update*** the representation with the new assets. For this operation, we ill perform an update (PUT) operation against the CDS webservice. With the zip file you created in step .9 above, run the following commandline

```
curl -X PUT -u %uname%:%passwd% -k -H "X-Requested-With: XMLHttpRequest" -F "File=@sdte400.zip" -H "Content-Type:multipart/form-data" %server%/ExperienceService/content/reps/sdte400
```

A helper batch script “create_cds_rep.bat” has been provided; this takes the zipfile name as a parameter.

```
> update_cds_rep sdte400.zip
```

2. Your repository should have PVZ and JSON files for both quadcopter models when complete. To check, you can run the following command

```
curl -u %uname%:%passwd% -k %server%/ExperienceService/content/reps
```

A helper batch script “list_reps.bat” file is provided.

```
> list_cds_reps
```

You should see something like this as the response

```
{"totalCount":1,"items":[{"name":"sdte400","createdby": "YOU","createdon":"DATE",
"modifiedby":"YOU”,"modifiedon":"DATE","url":"https://YOURSERVER/ExperienceService/content/reps/sdte400","metadata":{"version":"1.0.1","title":{"en":"Quadcopters"}, "description":{"en":" Models and
targets of various quadcopters for exercise 400"}}}]}
```

Note that the version number and description will have been updated to reflect the contents of the new version of metadata.json

3. Make a note of the url property – this is the location that our new representation is stored. We will use this in our experience.

```
"url":"https://YOURSERVER/ExperienceService/content/reps/sdte400"
```

402.3 Update Your Vuforia Studio Experience

Now that your configuration data has been stored inside ThingWorx, your Vuforia Studio experience needs to be edited to accept these changes.

1. Open `ScalingDigitalTwinExperiences402` in Vuforia Studio.
2. Add two new application parameters “guide” and “target”.
3. Add a 3dimage widget, and name it “photo”.
4. Set width to be 0.28 – this should match the US Letter sized image we want to load
5. Bind app parameter “guide” to the photo ‘resource’
6. Click to add a filter, in here enter the following

```
return '/ExperienceService/content/reps/sdte400/targets/' + value;
```

7. Delete the spatial target
8. Next, create a tml widget. Name it dynamicImageTarget

   The tml widget allows the programmer to create content at the DOM level.
   In this example, we are going to create an image tracking target
9. Click on the ‘edit’ button top open the TML widget edit panel. In this panel, enter the following:-

```
<twx-dt-target
  id = "dynamicimagetarget"
  guide-src = "{{app.params.guidesrc}}"
  src = "{{app.params.imagesrc}}"
  x = "0" y = "0" z = "0"
  size = “0.28”
```

```
  rx = "-90" ry = "0" rz = "0"
  istracked = "false" >
</twx-dt-target>
```

What is this doing? Well, we are declaring an AR tracking target; the database describing this target (src) is located at {{app.params.imagesrc}}, and the guide-view is {{app,params.guidesrc}}. The {{variable}} syntax says “replace the value of {{variable}} here”. We will see where this variable comes from shortly.

This target is what we refer to as a dynamic target – the target itself is a parameter to the experience, defined and stored elsewhere. You can follow the same pattern for other target types, e.g. a model target or an area target. The syntax it exactly the same – the target src (a URL) declares HOW and WHERE the target is defined.

The position and rotational values define the location of this tracking target relative to the original of our world. For dynamic examples like this, its best to keep the positional offsets (x,y,z) at zero. In this example, we’ve set a rotational offset of -90 degrees, which places the target horizontal, like a box on a tabletop.

10. Open the javascript window. Add the following :-

```
$scope.app.params.guidesrc =
  '/ExperienceService/content/reps/sdte400/targets/' + $scope.app.params.guide;
$scope.app.params.imagesrc =
  'vuforia-image:////ExperienceService/content/reps/sdte400/targets/' + $scope.app.params.target;
```

The first line creates a new application parameter (guidesrc) which is built up from a path (the location of the image in the CDS, the bundle that we uploaded earlier) appended with the {{value}} of the ‘guide’ parameter, which is passed to the experience when the thingmark is scanned and the code is resolved in the IRS.

The second line is similar, but in this example it creates a URI that references an image target. The URI protocol (vuforia-image:///) defines what the content is – an image target in this example - and the path declares where it can be found. Again, the value of the {{target}} application parameter is used here to form the final URI.

When the experience is launched, the act of scanning the thingmark results in the IRS resolving these different parameters, which are then handed to the experience as it starts. The code above the completes the task of defining the values that get inserted into the target definition (see step 8)

```
guide-src = "{{app.params.guidesrc}}"
 src = "{{app.params.imagesrc}}"
```

11. Next, we are going to display the new guideview. Studio has already provide an HTML template in the published application code. The template is, by default, left blank, but we can leverage this to display our guide image. Within the template HTML there is a that is declared with a class name “targetGuide”. Using HTML API, we can request the location of this div

```
let targetGuideDiv = document.querySelector("div.targetGuide");
```

if defined, we can then set the HTML ‘style’ for this div to show the image within the div.

```
if (targetGuideDiv) {
 targetGuideDiv.style.backgroundImage = "url('" + $scope.app.params.guidesrc + "')";
}
```

12. The final step is to change the model reference. In previous examples, we supplied an index value (1,2,3) and used this to build a fixed name for each model. To achieve this, we had used the data binding principal to link (bind) an application parameter (model) to the src property of a model widget. We assigned a ‘filter’ to this binding, with the filter reading

```
return '/ExperienceService/content/reps/sdte400/models/QuadcopterDT'+value+'.pvz';
```

In this example, we're taking this a step further by passing in the full name of the model. This means we can now work with any model. Let's make a small change to the filter :-

```
return '/ExperienceService/content/reps/sdte400/models/'+value;
```

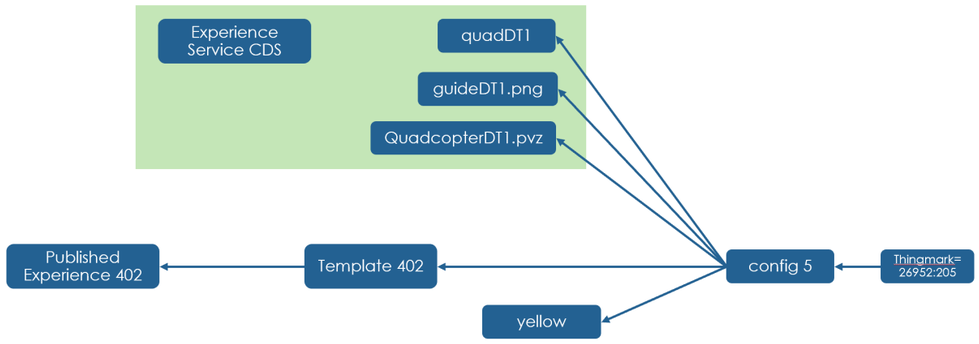
13. Your Vuforia Studio experience has now been updated to its proper state. Click **Publish** to publish your updated experience.
14. A Vuforia Studio project with the new changes added in this section named `ScalingDigitalTwinExperiences402` can be found in GitHub. **Note**: As with the last tutorial, this project is meant to be used as a reference material for the project file unless you have changed your template mapping.

402.4 Update the IRS

Next, we point our thingmarks at the new image targets.

In the previous example (401) we mapped various configs (config:1, config:2 etc.) to our template (template:401). These configs were already mapped to thingmarks that were introduced in earlier examples (202).

In this example, we will create two new configs, 5&6, and we will map these to two new thing codes. Here's an example showing the mapping for config:5.



As with other examples, a convenient batch script file is provided which includes all the settings; the text below describes what we are doing inside this batch script.

1. Lets start by defining a certain configuration – this config (config:5) will reference our QuadDT1 model, with a yellow painted shell. To do this, we create a key (urn:curriculum:config:5) and we map this to the various property values; color yellow, image guideDT1, target quadDT1 and model QuadcopterDT1.

   ```
   @curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:curriculum:config:5\", \"value\": \"urn:curriculum:color:yellow\"}" %server%/ExperienceService/id-resolution/mappings

   @curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:curriculum:config:5\", \"value\": \"urn:curriculum:guide:quadDT1/guideDT1.png\"}" %server%/ExperienceService/id-resolution/mappings
   ```

   Note that when referencing a target – Vuforia supplies a pair of files, target.xml and target.dat – the target reference should not include the file extension. Instead, we include an optional id parameter – if left blank, this says use whatever target is in this file. Vuforia allows multiple named targets in a single target database, so you can use this ID to index one of many. In this example, we'll leave it as the default.

```
@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:curriculum:config:5\", \"value\": \"urn:curriculum:target:quadDT1/quadDT1?id=\"}" %server%/ExperienceService/id-resolution/mappings
```

Finally, the model pvz :-

```
@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:curriculum:config:5\", \"value\": \"urn:curriculum:model:QuadcopterDT1.pvz\"}" %server%/ExperienceService/id-resolution/mappings
```

2. Next, we map config:5 to a new template; the template lets us share configs – you'll see in the script that we follow the same steps above for config:6.

```
@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:curriculum:config:5\", \"value\": \"urn:curriculum:template:402\"}" %server%/ExperienceService/id-resolution/mappings
```

4. And map the template to the experience, passing the parameters in the query section of the url.

```
@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{ \"key\":\"urn:curriculum:template:402\", \"value\":\"projects/scalingdigitaltwinexperiences402/index.html?expId=1^&target=^%7B^%7Bcurriculum:target^%7D^%7D^&model=^%7B^%7Bcurriculum:model^%7D^%7D^&vumark=^%7B^%7Bvuforia:vumark^%7D^%7D^&guide=^%7B^%7Bcurriculum:guide^%7D^%7D^&color=^%7B^%7Bcurriculum:color^%7D^%7D\", \"resourcetype\":\"Experience\",\"title\" : { \"en\":\"ScalingDigitalTwinExperiences402\" }, \"requires\" : [ \"AR-tracking\",\"w320dp\"  ], \"description\":{ \"en\":\"Curriculum demo 402\" } }" %server%/ExperienceService/id-resolution/mappings
```

:-

5. And finally, map a thingmark to our config

```
@set vumark5=<your thingmark>
@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:vuforia:vumark:%vumark5%\", \"value\": \"urn:curriculum:config:5\"}" %server%/ExperienceService/id-resolution/mappings
```

6. To test the IRS settings, run the **resolve** script (from previous examples)

   - **resolve.bat urn:vuforia:vumark:%vumark5%**

7. Repeat steps 1..6 for config:6. The batch file provided shows some examples.

402.5 Run your experience

With our IRS mapping complete, out experience published, we can test the full experience out by scanning one of the thinkmarks that you used in the step above.

402.6 Further investigation

Using a GS1 barcode instead of a thingmark….

<this could be delivered as a separate tutorial - TBC>

A recent capability added to View is the ability to utilize industry-standard GS1 format barcodes. These are the codes you find in the side of the packaging of a product. Here's an example EAN barcode (left) and the UPC-A equivalent on the right. UPC-A is the format used exclusively within the USA.

GS1 barcodes can be mapped to an Electronic Product Code (EPC) urn format. If view sees a GS1 compatible barcode, it will ask the IRS if there is a mapping to an experience, just as it does for Thingmarks.

Add the following to your IRS mapping

```
@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:epc:id:sgtin:0000000.004025\", \"value\": \"urn:curriculum:config:5\"}"
%server%/ExperienceService/id-resolution/mappings

@curl -u %uname%:%passwd% -H "Content-Type: application/json" -H "X-Requested-With: XMLHttpRequest" -k -d "{\"key\": \"urn:epc:id:sgtin:0000000.004026\", \"value\": \"urn:curriculum:config:6\"}"
%server%/ExperienceService/id-resolution/mappings
```

Once your mapping is complete, you can print out either/both of the imgDT1 / imgDT2 that we created back in 402.1.6 , scan the barcode and the experience will be launched with the correct model and color.