TECH NOTE

HOW-TO GUIDE

How to integrate barcode scanning in your Perspective project

This Technical Note contains all the information required to integrate barcode scanning into any Ignition Perspective project.

19/04/2022 – Ver. 1.0.0

KNOW YOUR LIMITLESS

TOC

1.Introduction

Ignition is an industrial application platform for collecting data, designing, and deploying industrial applications throughout an enterprise. It empowers the creation of any kind of industrial application including SCADA, MES, IIoT, reporting, alarming and more.

Perspective is the next generation visualization system for industrial applications, optimized specifically for mobile devices. Perspective puts the power of your plant floor in the palm of your hand by empowering you to create mobile-responsive industrial applications that run natively on any mobile device and web browser. The Perspective module has full HMI and SCADA capabilities and marks the beginning of truly mobile-optimized, touch-responsive, easily accessible applications for monitoring, control, analysis and data gathering in industrial systems.

Perspective is mobile-responsive, and that includes the use of mobile accessibility features like the device's GPS, accelerometer, camera, touch gestures and in this case, the barcode scanner.

There are 3 main topics that will be covered in this technical note:

1. The Barcode Scanner component
2. Demonstration
3. Working with barcode data

These topics are described in greater detail in the following sections

2. The Barcode Scanner component

The Barcode Scanner component is used to catch and covert the data from a barcode and to enable you to manipulate the data as needed. The following description is taken from the **Ignition 8.1 User Manual**:

The Barcode Scanner Input component awaits for input from a barcode scanner. The component was designed for keyboard wedge scanners, as the component provides dedicated prefix and suffix properties to define scanner input. As such, it can be useful to think of the Barcode Scanner Input component as a specialised text field that does not require focus, and uses characters to decide when to accepts and reject text input.

So how exactly does it work?

The scanner component is continuously listening, waiting for either the prefix and suffix to be entered, or the regex pattern to find a match. Once triggered, the component will load the scanned barcode string (excluding the prefix and suffix) into the data property for processing. The regex property can be used to extract specific fields from a scan, or validate data from the scan.

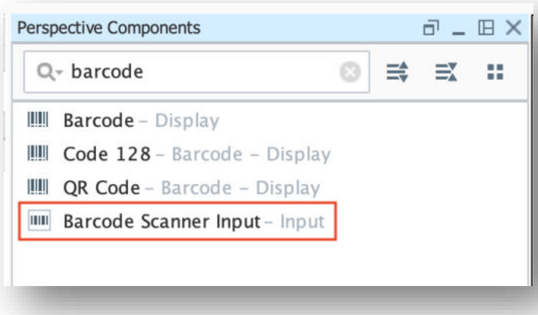
3. Demonstration

To demonstrate how to integrate barcode scanning into a project, we will be starting off with a blank project called Barcode.

You will need the Perspective module as the barcode scanning functionality gets activated by mobile devices' cameras.

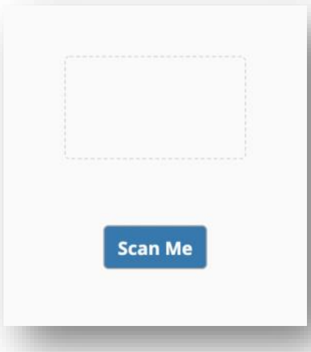
Let's run through the setup step by step:

1. Firstly, we need to tell the mobile device to launch the camera and be able to scan the barcode and send back the data associated with it. Luckily, Ignition has a component that does all the hard work for you, called the *Barcode Scanner Input*.

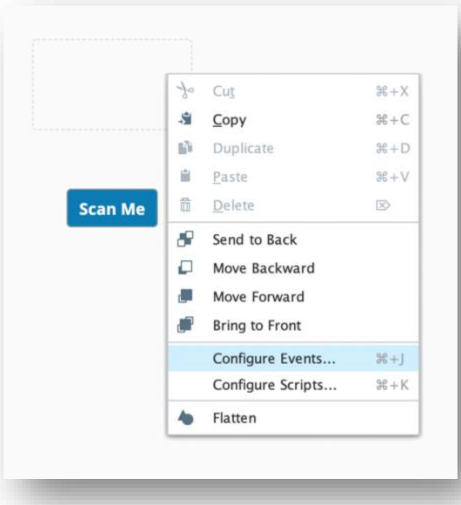

Perspective Components
barcode
Barcode – Display
Code 128 – Barcode – Display
QR Code – Barcode – Display
Barcode Scanner Input – Input

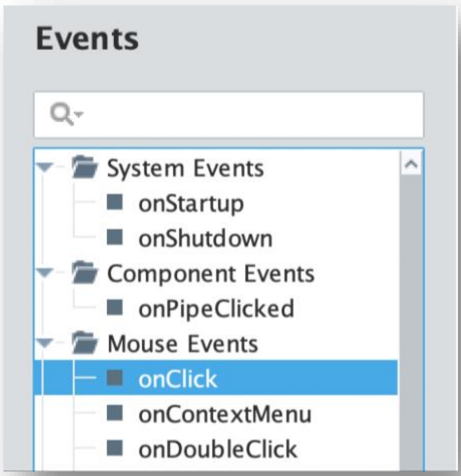
2. Take note, the following functionality you can also add to components that requires user interaction, for example, a button that the user needs to click

on to open the scanning functionality. Now we have a blank scanner component and a button called Scan Me:

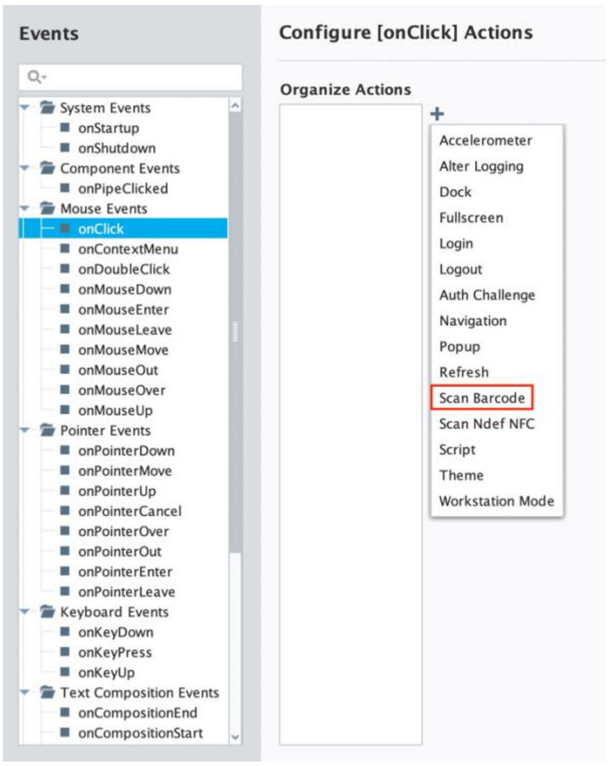


3. Right-click on one of the components, in this case I'll be using the scanner component, and click on Configure Events. We are using the onClick event for this demo:

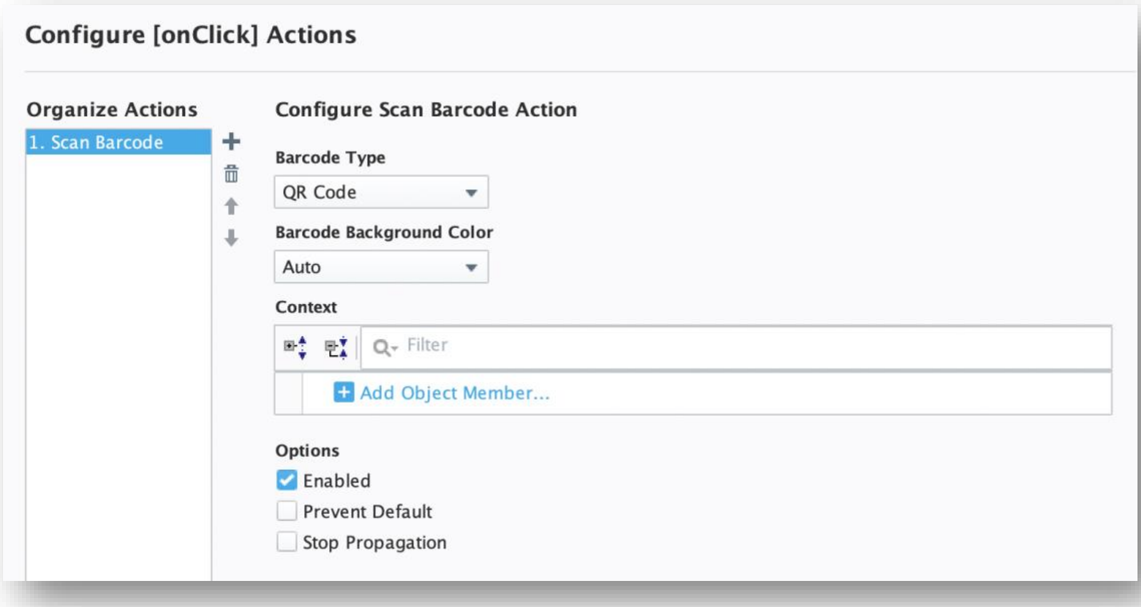




4. Next is to choose an action for this event, which is the Scan Barcode action. Click on the addition sign, and choose Scan Barcode:



5. You have the option to choose between different types of barcodes, in our case we'll be using the QR code. You also have the option to specify the barcode background color and add additional context, but we're going to keep it simple:


Configure [onClick] Actions

Organize Actions

1. Scan Barcode

Configure Scan Barcode Action

Barcode Type: QR Code

Barcode Background Color: Auto

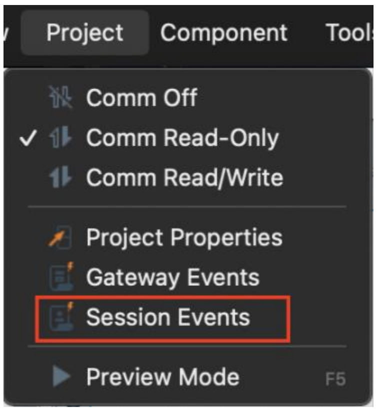
Context

Options

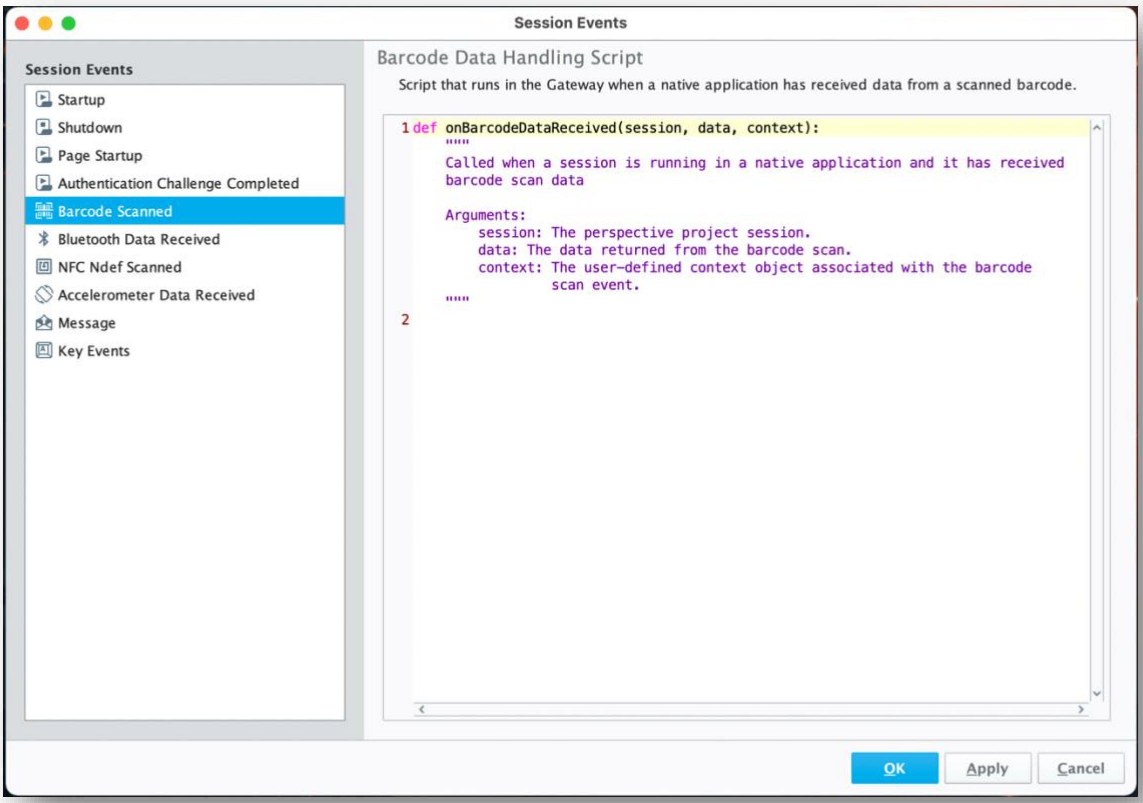
- [x] Enabled
- [ ] Prevent Default
- [ ] Stop Propagation

6. After clicking on OK, this will physically launch the scanning feature on the device when the component is clicked on. Next, we need to tell the session what to do with the barcode data once it has been scanned.

7. Because we are using perspective, and it works in Sessions, we will be creating a session event to tell Ignition what to do with the barcode data. Click on Project in the toolbar and choose Session Events:



8. Choose the Barcode Scanned event. You'll see it opens a script that can receive the data as well as additional contextual data if chosen:



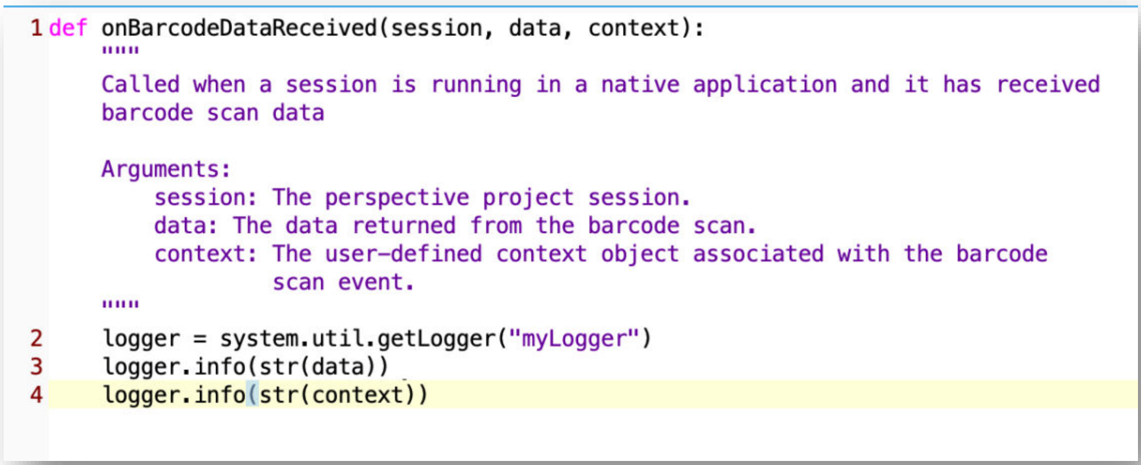
Barcode Data Handling Script

Script that runs in the Gateway when a native application has received data from a scanned barcode.

```
def onBarcodeDataReceived(session, data, context):
	"""
	Called when a session is running in a native application and it has received
	barcode scan data

	Arguments:
		session: The perspective project session.
		data: The data returned from the barcode scan.
		context: The user-defined context object associated with the barcode
		         scan event.
	"""
```

9. Firstly, we want to log the data so that we can see what it looks like. This will help to understand how the data is received and find the best way to store, manipulate and display the data afterwards. For that, we want to create a logger and log the data and context parameters:

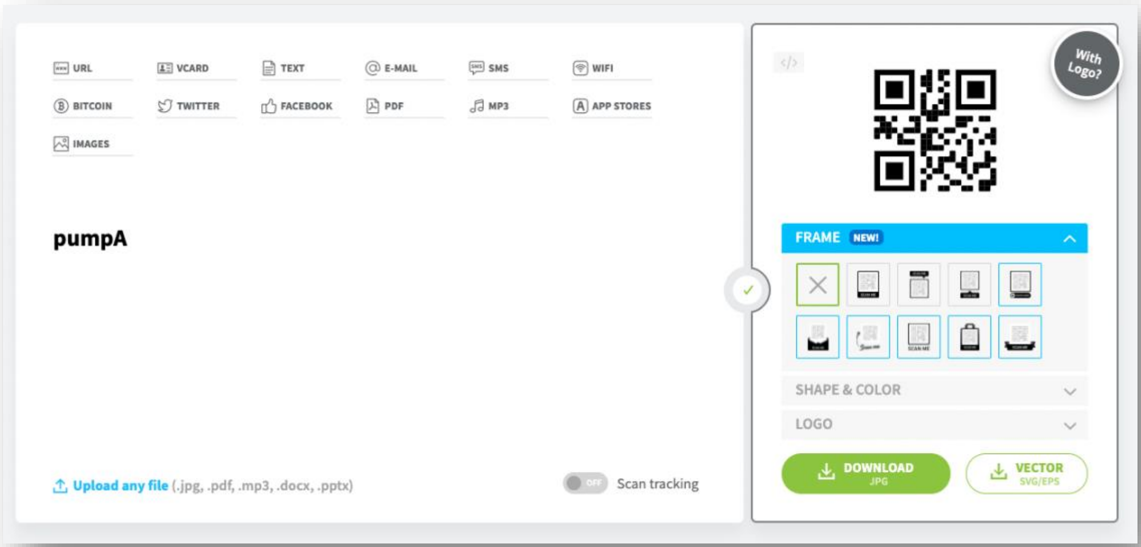


```
def onBarcodeDataReceived(session, data, context):
	"""
	Called when a session is running in a native application and it has received
	barcode scan data

	Arguments:
		session: The perspective project session.
		data: The data returned from the barcode scan.
		context: The user-defined context object associated with the barcode
		         scan event.
	"""
	logger = system.util.getLogger("myLogger")
	logger.info(str(data))
	logger.info(str(context))
```

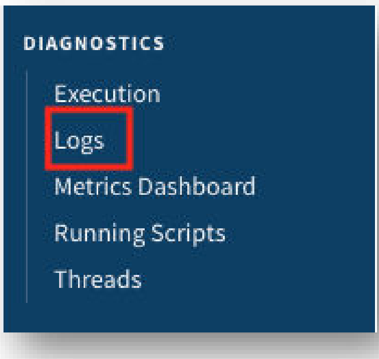
10. The next step is to open the project as an application on the Perspective app. You can click on the button or component to scan a QR barcode.

11. We created a QR code with some text referring to 'pumpA'. If you want to test it out with a dummy barcode, you can go to https://www.qr-code-generator.com/ and create your own. You can also enter a website URL or drop files into the edit field for the QR code:

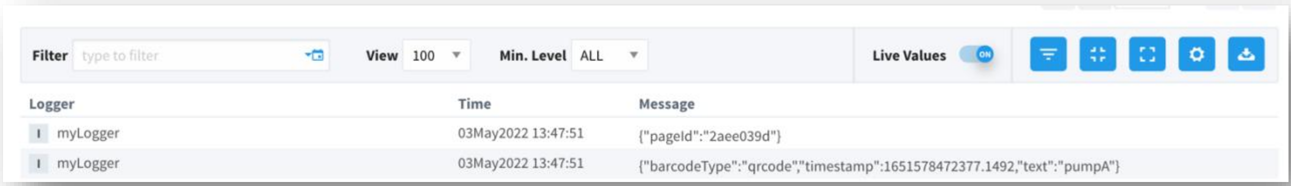


pumpA

12. After scanning it with a mobile device, that logger we created will log the data and context data to the Logs in the Gateway. Navigate to your Gateway, under the Status Tab find Logs under Diagnostics:



13. There are 2 lines of logs we want to see, the first line where we logged the data and the second line where we logged additional context data. You'll see that for my 'text' field, it logged the 'pumpA' text we added into the barcode. You can also see data like the barcode type, timestamp, and page id:



Logger	Time	Message
I myLogger	03May2022 13:47:51	{"pageId":"2aee039d"}
I myLogger	03May2022 13:47:51	{"barcodeType":"qrcode","timestamp":1651578472377.1492,"text":"pumpA"}

14. With this data, you can start using barcode scanning to navigate to different pages in the application, write values to tags or down to plc's, or even just save the barcode data to a database.

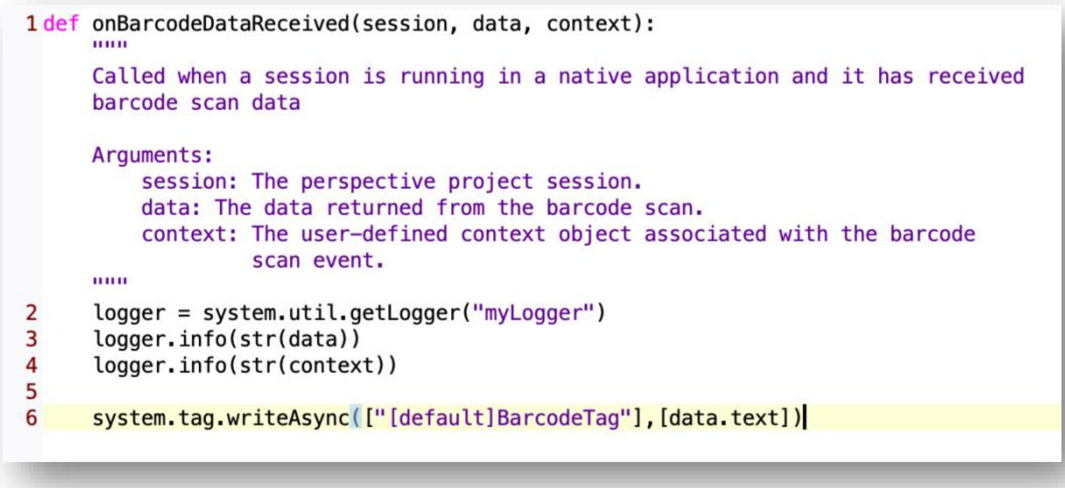
4. Working with barcode data

Writing to a tag:

1. For this, you'll need a standard memory tag of type string which we'll be writing to. You can also write to other types of tags if you know how to convert the data types correctly OR how to correctly write to a PLC tag with permission.

2. Opening the session events, we'll be adding a line of code to write the data to the tag:

   ```
   system.tag.writeAsync([“[default]TagName”],[data.text])
   ```

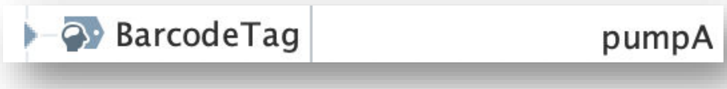


```
def onBarcodeDataReceived(session, data, context):
    """
    Called when a session is running in a native application and it has received
    barcode scan data

    Arguments:
        session: The perspective project session.
        data: The data returned from the barcode scan.
        context: The user-defined context object associated with the barcode
                 scan event.
    """
    logger = system.util.getLogger("myLogger")
    logger.info(str(data))
    logger.info(str(context))

    system.tag.writeAsync(["[default]BarcodeTag"],[data.text])
```

3. When you scan the barcode again, you'll see the text 'pumpA' gets saved as the tags value:



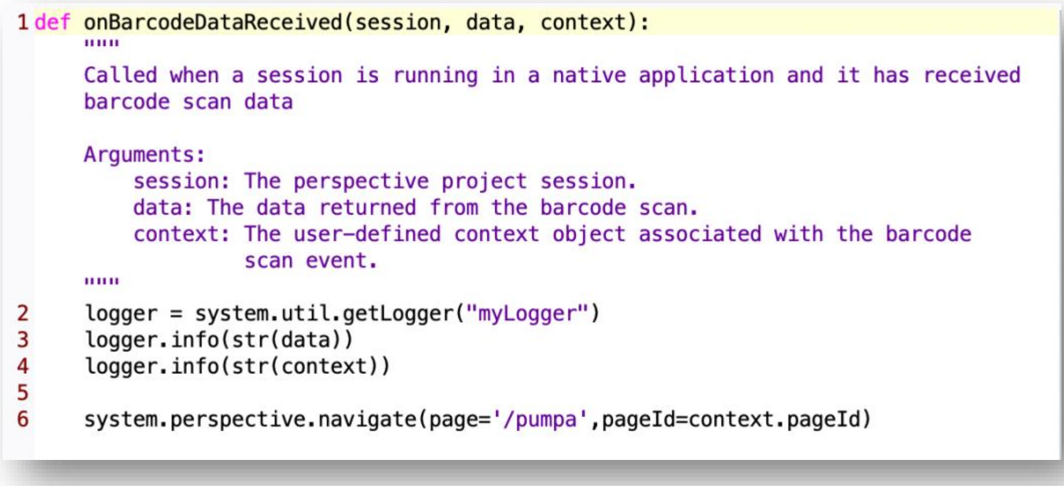
BarcodeTag pumpA

Navigating to another page:

1. For this, you'll need more than one view to which you want to navigate, and we'll just be navigating to another view called pumpA.

2. Opening the session events, we'll be adding a line of code to use the context data of the barcode (pageId) to navigate to page pumpA:

   system.perspective.navigate(page='/pumpa', pageId=context.pageId)



```
def onBarcodeDataReceived(session, data, context):
    """
    Called when a session is running in a native application and it has received
    barcode scan data

    Arguments:
        session: The perspective project session.
        data: The data returned from the barcode scan.
        context: The user-defined context object associated with the barcode
                 scan event.
    """
    logger = system.util.getLogger("myLogger")
    logger.info(str(data))
    logger.info(str(context))

    system.perspective.navigate(page='/pumpa',pageId=context.pageId)
```

3. Once you have scanned the barcode, the application should switch to the page you wanted to navigate to.

5. Additional Resources

How to integrate barcode scanning in your Perspective project

There is a demonstration at 41:27 in the video on how barcode scanning is used to get product information from scanning the barcode from a book.