iDimension

QubeVu
Version 1.0

API GUIDE

RICE LAKE®
WEIGHING SYSTEMS

PN 167741 Rev A

1 Table of Contents

2 Introduction

2.1 Purpose

This document specifies how a QubeVu can be connected to a user's client system and how that system can then obtain information from the QubeVu about an item's dimensions and the item's image.

2.2 Scope

This document describes only the web service interface to QubeVu. The web service interface is not supported on QubeVu prior to QubeVu Mark 4.

Only those interfaces that support dimensioning and image retrieval are discussed here.

3 Background

QubeVu provides a solution to the problem of quickly and reliably measuring the three dimensions of a mail item as part of the mail acceptance or handling process. It also provides high resolution images of the top surface of the item that can subsequently be used for various purposes including barcode recognition and OCR or ICR[1].

It often operates in conjunction with a weigh scale that is collecting the weight of the item and the QubeVu is positioned centrally over the weigh scale so that the dimensions can be captured at the same time as the weight.

The QubeVu is able to automatically recognise that an item has been placed underneath it and begin the process of measurement immediately. Hence this information can already be available by the time the user's client system requires it.

On installation the QubeVu is calibrated to suit the location and the specific characteristics of the individual device.

Configuration data for the QubeVu is held in files on its embedded data storage.

[1] OCR or ICR capability requires a separate Leadtools 15 license

4 Overview

The QubeVu connects to the user's workstation with a single Ethernet connector.

The QubeVu requires one power socket.

No specialised software components or drivers are installed on the workstation. All necessary software components are embedded in the device. Client applications can interface with the device via a web service interface. The web service starts automatically when the device is started.

The interface components defined in this document provide the mechanism by which the client application can communicate with the QubeVu.

The client processing is started by the user in response to the package being presented by the customer and after the item has been placed under the QubeVu.

There is a separation between the client support interface and the internal operation of the QubeVu software within the service. The client interfaces read what information is currently available from the QubeVu and report the status of the information. These calls do not block waiting for the required information. This is to prevent the client interface hanging in indeterminate circumstances.

The client application will therefore probably want to implement some form of delayed loop to read the QubeVu information by monitoring the status on the interface it is calling. It may also implement its own time-out to prevent the UI hanging indefinitely if there is a problem.

The web service can be found at the following url: http://{device}/WebServices/QubeVuService.

The web service supports the HTTPPost binding. Client must use this binding to interact with the service. A C# proxy class (QubeVuServiceHttpPostClient.cs) for the service is available as part of the API. Other proxy classes may be generated if necessary from the WSDL file. The WSDL file of this service can be found at http://{device} /WebServices/QubeVuService. A second C# proxy class (QubeVuServiceAsyncClient.cs) is available for use with Windows Store apps and Universal Windows Platform development projects (including, but not limited to Windows IoT).

5 Interface Specifications

The interfaces are implemented within the ***QubeVuService*** web service.

Information is only available while an item is on the QubeVu and the QubeVu has had time to process it. Once the item is removed the information is no longer available.

Status information, including information about the mail item, is retrieved through the ***Status*** method. The ***Status*** method only returns a URL for any images captured. Image content is retrieved using standard HTTP requests.

An XML Error element with the error message will be returned in the event of a software error.

5.1 Retrieving the Dimensions

This uses the Status interface: ***QubeVuService/Status***.

It returns a single string value containing an XML document.

The actual content of the XML depends on whether the call is successful or not. This reflected in the Error element.

- If the call is successful then there is no Error element.
- If the call fails, then an error code and an error message are returned.
- There may be no data as the call is too soon after the item was placed under the QubeVu. In this case there will be no Dimensions element and the status attribute will indicate why. Dimensions are only available when status is set to IMAGING or REMOVE – see below for other values.

```
<QVStatus
     CaptureId="string"
     Status="string"
     ExtendedStatus="string"
     OutOfBounds="int"
>
  <ExternalData>
    <Barcode>
      <TextData>string</TextData>
    </Barcode>
  </ExternalData>
  <Error Code="int" Message="string">
  <CapturedData CaptureDefinitionName="string"> [Only present if available]
    <DateTime>string</DateTime>
    <Weight>float</Weight>
    <ScaleData>
      <Error Code="int" Message="string">       [Only present if error]
      <Weight>int</Weight>
      <ScaleFactor>int</ScaleFactor>
      <IsStable>boolean</IsStable>
      <WeightUnit>string</WeightUnit>
      <RawData>string</RawData>
      <DisplayWeight>string</DisplayWeight>
    </ScaleData>
    <Dimensions                                 [Only present if available]
```

```
          Irregular="boolean"
          Undersize="int"
          Oversize="int"
          Refinement="int"
          DimUnit="string"
          OutOfBounds="int"
          UnknownDimensions="boolean"
          UnknownDimensionsReason="string">
        <Height>decimal</Height>
        <Length>decimal</Length>
        <Width>decimal</Width>
      </Dimensions>
      <TrackerImage />                          [Only present if available]
      <LowResImages>                            [Only present if available]
        <LowResImage />                         [Only present if available]
      </LowResImages>
      <HighResImages>                           [Only present if available]
        <HighResImage>                          [Only present if available]
          <Barcodes/>
        </HighResImage>
      </HighResImages>
    </CapturedData>
    <Crc>string</Crc>
</QVStatus>
```

This not a blocking call and may require the client to use a delayed loop to call it until it returns the required response.

/QVStatus	Top level wrapper element
./@CaptureId	Sequential capture identifier. CaptureId is incremented each time QubeVu processes an item and is reset when the system is restarted.
./@Status	The current status of the device/operation. See below for more details.
./@ExtendedStatus	Extended information about the status of the device/operation. See below for more details.
./@OutOfBounds	Flag indicating that an item extends out of the sensor's field of view. (1) Left, (2) Right, (4) Top, (8) Bottom or a combination of these.

/QVStatus/Error	Error details
./@Code	Error Code
./@Message	Error Message
/QVStatus/CapturedData	Data captured by QubeVu
./@CaptureDefinitionName	Name of capture definition that has triggered the capture
./@CaptureId	Sequential capture identifier. CaptureId is incremented each time QubeVu processes an item and is reset when the system is restarted.
./ExternalData/Barcode/TextData	The data read by the externally attached USB scanner. *(firmware 4.6.2 and higher)*
./DateTime	The date and time of the scan.
./ScaleData	Data reported from attached scale or received through Scale Service interface.
./Error	Scale-specific error details
./@Code	Error code
./@Message	Error message
./Weight	The weight of the item as an integer. Use scale factor to find number of decimal places.
./ScaleFactor	Indicates the number of decimals in ./Weight
./IsStable	Indicates whether the weight is stable or not.
./WeightUnit	Specifies the unit in use for the weight.
./RawData	Hex encoded raw data as received from the scale.

./DisplayWeight	Weight and unit reported from the scale using the scale's own unit settings and display format.
/QVStatus/CapturedData/Dimensions	The dimensions of the item
./@Irregular	Irregular shaped object (true or false)
./@Undersize	Flag indicating undersize Height (4) or Width (2) or Length (1) or a combination of these.
./@Oversize	Flag indicating oversize Height (4) or Width (2) or Length (1) or a combination of these.
./@Refinement	Refinement of Width (2) or Length (1) or a combination of these. The Refinement field is a bit field indicating the refinement state of each dimension and whether the current certificate settings require the dimensions to be refined. bit1 := set if Length is refined bit2 := set if Width is refined bit3 := set if Height is refined bit4 := set if refinement is required (this is set under certification settings) If refinement is not required (i.e. bit4 is not set) then the other bits should be ignored. As an example a value of 8

	means that refinement is required and Length, Width and Height are refined. Note that for the LTL system, refinement is not required.
./@DimUnit	The unit of measure. Valid values are ‘in’, ‘mm’, ‘cm’ and ‘m’.
./@OutOfBounds	Flag indicating that an item extends out of the sensor’s field of view. (1) Left, (2) Right, (4) Top, (8) Bottom or a combination of these.
./@UnknownDimensions	When dimensions cannot not be determined the unit returns 0-s for length, width and height and this indicator is set to true. *(firmware 4.4.2 and higher)*
./@UnknownDimensionsReason	When dimensions cannot not be determined and UnknownDimensions is true, the UnknownDimensionsReason attribute provides additional information about why the system was not able to dimension the item. For possible values and explanations, please see end of this section. *(firmware 4.4.2 and higher)*
./Length	Length of object (longer dimension in the X,Y plane)
./Width	Width of object (shorter dimension in the X,Y plane)
./Height	Height of object (dimension in

	the Z plane)
/QVStatus/CapturedData/RawDimensions	Included if raw (unrounded) dimensions are enabled
./Length	Length of object (longer dimension in the X,Y plane)
./Width	Width of object (shorter dimension in the X,Y plane)
./Height	Height of object (dimension in the Z plane)
/QVStatus/CapturedData/LowResImages	Wrapper for zero or more LowResImage elements
/QVStatus/CapturedData/LowResImages/LowResImage	
./@Url	URL of low-res image
./@Name	
./ItemRect	Either this element or ItemWireframe is filled in.
./ItemRect/CenterX	
./ItemRect/CenterY	
./ItemRect/D1	
./ItemRect/D2	
./ItemRect/Theta	
./ItemWireframe	Either this element or ItemWireRect is filled in.
./ItemWireframe/Faces	Multiple faces may be defined
./ItemWireframe/Faces/Face	
./@Visible	“false” if specified face is obstructed from view, “true” if visible

./Vertices	Multiple points may be defined under this element
./Point	
./X	X pixel coordinate of the vertex
./Y	Y pixel coordinate of the vertex
/QVStatus/CapturedData/HighResImages	Wrapper for zero or more HighResImage elements
/QVStatus/CapturedData/HighResImages/HighResImage	
./@Url	URL of high-res image
./@Name	
./ItemRect	Either this element or ItemWireframe is filled in.
./ItemRect/CenterX	
./ItemRect/CenterY	
./ItemRect/D1	
./ItemRect/D2	
./ItemRect/Theta	
./ItemWireframe	Either this element or ItemWireRect is filled in.
./ItemWireframe/Faces	Multiple faces may be defined
./ItemWireframe/Faces/Face	
./@Visible	“false” if specified face is obstructed from view, “true” if visible
./Vertices	Multiple points may be defined under this element
./Point	
./X	X pixel coordinate of the vertex

./Y	Y pixel coordinate of the vertex
./Barcodes	Multiple Barcode elements, one for each barcode found, may appear under this item.
./Barcodes/Barcode	Details of a single barcode
./RawData	Barcode value (deprecated)
./EncodedData	Barcode value in hex encoded format
./DecodedData	Barcode value
./TextData	Barcode value
/QVStatus/CapturedData/TrackerImage	Details of low resolution image used for finding item
./@Url	
./@Name	
./ItemRect	Either this element or ItemWireframe is filled in
./ItemRect	Either this element or ItemWireframe is filled in.
./ItemRect/CenterX	
./ItemRect/CenterY	
./ItemRect/D1	
./ItemRect/D2	
./ItemRect/Theta	
./ItemWireframe	Either this element or ItemWireRect is filled in.
./ItemWireframe/Faces	Multiple faces may be defined
./ItemWireframe/Faces/Face	
./@Visible	“false” if specified face is obstructed from view, “true” if visible

./Vertices	Multiple points may be defined under this element
./Point	
./X	
./Y	
/QVStatus/Crc	Hex string of CRC32 checksum generated over the entire response string prior to inserting the CRC element. To validate checksum, calculate CRC32 over the entire response as it was received but exclude the CRC element and then compare to value stored in this element. Note that checksum includes all characters including whitespace and therefore cannot be calculated on an xml string that has been reconstructed by an xml parser from the parsed nodes.

Error codes and message:

Error	
Code	**Message**
0	None.
1	Hardware Initialization FAILED.
2	Tracker Config Initialization FAILED.
3	Missing RegistrationMarksCropped.bmp.
4	Setting reference image for Targetfinder FAILED.
5	Loading of Calibration files FAILED.
6	Getting new Images from hardware FAILED.
7	Tracking FAILED.
8	Calibrating.
9	TCP Server Port binding failed
10	TCP Server exception in Processing Client
11	TCP Server time out on Imaging

12	Low res camera needs to be calibrated first!
13	Calibration stopped.
14	Error loading / parsing Configuration.
15	Unable to save Calibration to file.
16	Unable to use name set in Capture/Get Command. CaptureDefinition with name were not set.
17	Invalid CaptureDefinition command.
18	Unable to delete Calibration files.
19	Unable to Zero Height
20	Failed to write or verify audit trail

Scale-specific error codes and messages:

Error	
Code	**Message**
0	None.
1	Time out waiting for stable weight.

UnknownDimensionReason codes and explanation:

UnknownDimensionReason	
Code	**Explanation**
<blank>	Dimensions determination successful. (UnknownDimensions attribute is set to “false”)
Timeout	Dimensions could not be established within system specified timeout period.
InvalidBackground	Dimensions could not be established because item-free background could not be acquired. Please clear the platform, zero the height and try to scan again.
InvalidBackgroundOrOutOfBounds	Dimensions could not be established because object is out of bounds or item-free background could not be acquired. Please clear the platform, zero the height and try to scan again by placing the item completely within the scan zone.
NoDimItem	Dimensions could not be established because system is configured not to measure this type of item.
InvalidDepthData	Dimensions could not be established due to an unexpected inconsistency in the measured depth data.
Unknown	Dimensions could not be established; reason not given.

5.2 Retrieving the Image.

In designing the client dialogue that uses the QubeVu it should be kept in mind that the QubeVu image is available some time later than the dimensions as it has to determine the dimensions before it can

adjust the camera to take the image(s). The transfer time of the images to the workstation also takes a finite time.

The image returned is of the top face of item. The high resolution camera will zoom in according to the parameters specified in the capture definition.

This uses HTTP GET: ***GET {image url} HTTP/1.1***

The image url is retrieved from the response of a successful Status call as described above. The image is only available if the status value is set to REMOVE.

It returns an image file in bitmap format. The colour depth and resolution of the image depends on the parameters specified in the capture definition.

5.3 Setting zero height of scan platform

The QubeVu device records the distance between the camera and the plane of the baseplate during calibration. The base platform height changes when a scale, which rests on the baseplate, is attached or removed from the QubeVu setup. When such a change in base platform height occurs it is necessary to tell the device to measure the height of the distance between the base and the camera anew. This process is referred to as "zeroing height". Customers must perform a zero height operation when the effective platform is raised or lowered for any reason or the device will likely not scan items until this operation is successfully performed. (This API call does not require QubeVu tracker to be restarted.)

Zeroing height on the QubeVu device involves placing the calibration object on the platform, stepping back and then making the API call below with "true" passed in as parameter. If the platform has a completely smooth, uniform surface, then it is not necessary to use a calibration object. If the surface has any bumps, ridges, a scale with textured top plate or it has rollers then a calibration object must be placed on the platform before calling SetZeroHeight. Only when the base plane is completely smooth and flat (a dot or graphic or pattern is OK as long as it's flat and level with the baseplate) then you can call the API with the "false" argument and NOT use a calibration object for the zero height process.

Method signature:

```
QVServiceResponse SetZeroHeight(bool usingCalibrationObject)
```

The process usually takes 1-6 seconds. During this time the device should be left undisturbed. The method returns a single string value containing an XML document:

```
<QVServiceResponse>
  <Error Code="int" Message="string" >
</QVServiceResponse>
```

At this point the calibration object may be removed from the platform (if one was used). Error Code 0 indicates a successful zero height outcome. Any error code other than 0 should be interpreted as failure to zeroing height. It is recommended that implementers of the zero height process offer users a chance to retry zeroing the height upon failure. A retry simply means another call to SetZeroHeight as before.

5.3.1 Zero height error messages

Message	Meaning and Resolution

Motion detected	Motion was detected during the zero height process. Please make sure to not interfere with the device during zero height operation. Leave the platform clear of objects completely or leave the platform with just the calibration object placed on top of the scale (if one is used) or other non-smooth surface.
Height not stable	Please make sure to not interfere with the device during zero height operation and try zeroing height again. If problem persists please contact Postea support.
Change in height exceeded set value	There was an unexpectedly large difference in previously known height and newly measured height. To remedy this situation please make sure that the new height is closer to the originally measured height or alternatively, you may increase the *Zero height max change* (ZeroHeightChangeMaxMM) configuration item in Admin tools → Measurement Settings.

5.4 Retrieving barcodes

Reading barcodes requires the capture of an image by the high resolution camera typically at 140dpi or better but the actual resolution will depend on the barcode to be recognized. QubeVu provides two approaches to retrieve barcodes from the captured high resolution images.

One approach is to simply specify the types of barcodes of interest in the capture definition. Once this is setup then QubeVu will search for barcodes in the corresponding high resolution images and will return the values of each barcode found in response to a Status along with the high resolution images' urls. The found barcodes are returned in the Barcodes element of each HighResImage.

Another approach is to use the QubeVuBarcodes Client API. This API can be used by the client application to find barcodes from a high resolution image captured by QubeVu. The API is simple to use and is described in detail in Appendix B.

The benefit of using the client API is that it can take advantage of the fast processors of the client hardware and get the results faster.

Note that the number of barcode types enabled will have an effect on performance; the more barcode types that are enabled the longer it will take to process an image. For optimum performance minimize the number of barcode types that are enabled.

5.4.1 Enabling barcode recognition through capture definition

Use the CreateCaptureDefinition interface to enable barcode recognition.

E.g.

```
<CaptureDefinitionDetail Name=\"autotriggerparcel\">
     <TimeoutMsecs>0</TimeoutMsecs>
     <NoDimItems>Flat</NoDimItems>
     <LowResImages></LowResImages>
     <HighResImages>
          <HighResCamCapture ImageName=\"HighResImage1\">
               <MinDpi>140</MinDpi>
               <MaxDpi>140</MaxDpi>
               <BarcodeCapture>
                    <BCTypes>CODE128 CODE93 CODE39</BCTypes>
               </BarcodeCapture>
```

```
            </HighResCamCapture>
        </HighResImages>
</CaptureDefinitionDetail>
```

For information on the CreateCaptureDefinition interface please see the Capture Definitions section below.

5.5 Setting the weight from an external scale.

In a typical configuration a weighing scale is connected to QubeVu to detect item placement and removal. In some situations it may be necessary to have the scale connected to the client computer directly rather than to the QubeVu device. One such scenario is where there is an existing client application that requires direct control of the scale. Operating QubeVu with an external scale requires that the 'Scale type' configuration item be set to 'EXTERNAL' and that a process on the client computer monitors the scale and notifies QubeVu of any weight changes. It is important that the weight change notification happens in a timely fashion to ensure that it is in synch with what the scan head sees.

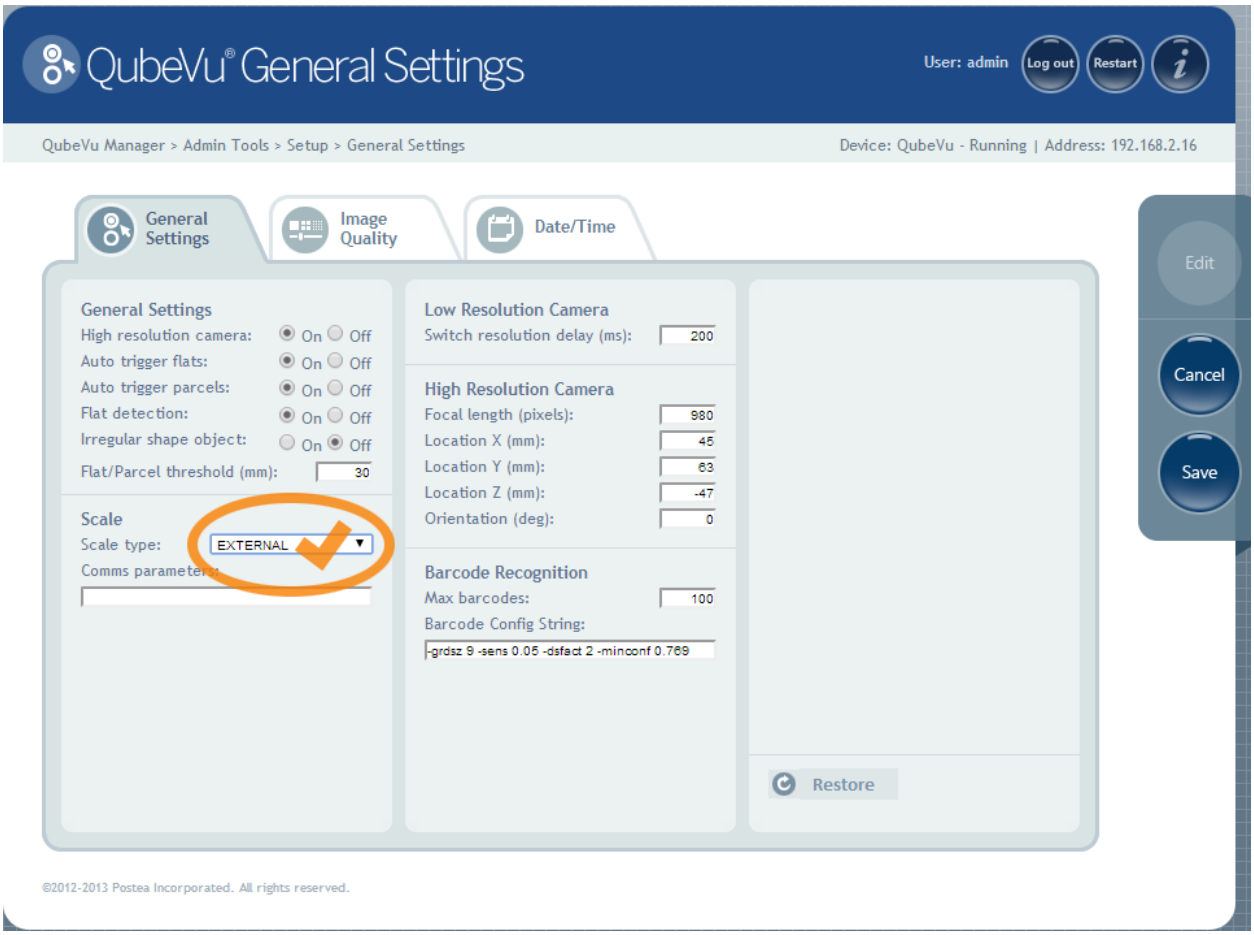


Figure 1 - External scale configuration

Client application must use the QubeVu's scale service to notify the device of any weight change when using the external scale configuration. The scale web service can be found at the following url: http://{device}/WebServices/ScaleService.

Use the **ScaleService/SetCurrentWeight** web service method to notify QubeVu of any weight changes, regardless of whether the weight is zero or not or the weight is stable or not.

It is important to note that when the QubeVu first starts up it will not enter the 'Ready' state until it has been notified that there is a stable zero weight being output from the scale. This will be automatically detected if the scale is connected directly to the QubeVu but must be communicated to the QubeVu via the **SetCurrentWeight** web service method when using an external scale configuration.

The best way to implement this is to use a background process/thread/task that constantly queries the current weight from the scale and then calls the **SetCurrentWeight** method with the values received from the scale.

Method signature:

```
QVServiceResponse SetCurrentWeight(int weight, int scaleFactor, bool isStable,
string weightUnit)
```

weight - The current weight returned by the scale.

scaleFactor – A positive of negative power of ten to be used to multiply the value in *weight* to get the actual weight in the specified units.

isStable – Indicates whether the specified weight reading is stable or not. A stable weight is required to start capture. A stable zero weight is required for background updates, however, a non-stable zero weight or drop in weight will signal item removal/or replacement.

weightUnit – The unit of the weight specified. Currently supported units are 'g' for grams and 'oz' for ounces.

The method returns a single string value containing an XML document.

```
<QVServiceResponse>
  <Error Code="int" Message="string" >          [Only present if error]
  <CapturedData CaptureDefinitionName="string"> [Only present if available]
</QVServiceResponse>
```

5.6 Returned values

5.6.1 Status

Status value	Meaning
STARTING	The service is starting up.
STARTED	The service has started but is not ready for processing. If the device is in this status for more than a couple of seconds after starting then it is very likely that there is an item on the platform that needs to be removed. The platform should be clear when the device/service is starting.
READY	The device is ready and waiting to be used – there is no item on it.

TRACKING	The device is processing a change in image after an item has been placed under it.
IMAGING	The device measurements have been determined and the camera is being adjusted to take the image.
REMOVE	The image has been fully processed – the item can be removed when the client processing has completed.
WAIT	Preparing the device for the next item. The previous image and dimensions are deleted from memory and the lens is reset.
STOPPED	The service has stopped – there is some problem.
CALIBRATING	The device is in calibration mode.
CONFIGURING	The device is in configuration mode.

5.6.2 Extended status

The returned extended status attribute contains none or more comma separated strings that provide additional information about the status of the device.

Constraint value	**Meaning**
ScaleNotStable	This is set during tracking if the scale indicates that the value returned is not a stable value. This is only used when a recognized scale is connected to the system. Processing will not progress to the next step until this flag is cleared by receiving a stable weight from the scale.
MotionDetected	This is set during tracking and ready states and indicates that the system has detected movement. Processing will not progress to the next step while this is set.
ItemDetected	This is set when the system has detected that an item is placed on the device platform/scale. When a scale is used this indicates that weight returned is not zero. In 'scale-less' mode this indicates that the system cannot find the target panel.
ItemNotDetected	This is set when the system is in ready mode and there is no item on the platform/scale.
TrackerNotConfident	This indicates that the tracker detected an item but it is not confident what the dimensions of the item are. After a timeout (configurable) the system will progress to next step and return zero-valued dimensions.
ExceptionOccured	This is set when an exception occurs.
DeviceNotStable	This is set during tracking if one of the sensors indicates that the sensor value returned is not a stable value. Processing will not progress to the next step until this flag is cleared by receiving a stable value from the sensor.
ServiceStarting	This is set when the system is initializing.
ConfigMode	This is set when the system is in configuration mode, such as during calibration or image exposure adjustment. A restart operation takes the device out of configuration mode.

ResultNotStable	This is set when the item is being manipulated such as when the item is in the act of being placed on the platform or removed from it.
ItemOutOfBounds	This indicates that the item protrudes outside the measurable area. A repositioning of the item is necessary.
WaitingToWarmUp	This is set during the warmup period. If device is used in a certified-for-trade application the warm-up period must have been elapsed before certified measurements can be taken.
PlatformNotClear	This is set when there is something on the platform.

5.7 Capture definitions

5.7.1 Creating capture definitions

Use the CreateCaptureDefinition interface: ***QubeVuService/CreateCaptureDefinition***

Method signature:

```
QVServiceResponse CreateCaptureDefinition(string name, string definitionString)
```

name - The name of the capture definition to be created

definitionString – The XML string describing the capture definition.

```
<CaptureDefinitionDetail Name="string" >
     <TimeoutMsecs>int</TimeoutMsecs>
     <NoDimItems>nodim_options</NoDimItems>
     <LowResImages>
          <LowResCamCapture ImageName="string" >
               <ResX>int</ResX><ResY>int</ResY>
               <Markings>
                    <SerialNumber Visible="boolean"/>
                    <DateTimeStamp Visible="boolean"/>
                    <ScanId Visible="boolean"/>
                    <Dimensions Visible="boolean"/>
                    <Indicators Visible="boolean"/>
                    <ItemOutline Visible="boolean"/>
               </Markings>
          </LowResCamCapture>
     </LowResImages>
     <HighResImages>
          <HighResCamCapture ImageName="string">
               <MinDpi>int</MinDpi>
               <MaxDpi>int</MaxDpi>
               <BarcodeCapture>
                    <BCTypes>bctype_list</BCTypes>
               </BarcodeCapture>
          </HighResCamCapture>
     </HighResImages>
</CaptureDefinitionDetail>
```

The method returns a single string value containing an XML document.

```
<QVServiceResponse>
  <Error Code="int" Message="string" >          [Only present if error]
  <CapturedData CaptureDefinitionName="string"> [Only present if available]
</QVServiceResponse>
```

/CaptureDefinitionDetail	Top level wrapper element
./@Name	Name of the capture definition.
./TimeoutMsecs	The system will report a timeout when it is unable to detect an item within the time specified by the capture definition. If the timeout is set to zero, then system will continue trying indefinitely. *(firmware 4.4.2 and higher)*
./NoDimItems	Specifies the type of items that should be processed without dimensioning. For LTL systems this must be set to 'None'. See table below for more options.
/CaptureDefinitionDetail/LowResImages/LowResCamCapture	Low resolution image parameters
./ResX	Specifies the X resolution of the low resolution image to be taken. For LTL sytems this should be 640.
./ResY	Specifies the Y resolution of the low resolution image to be taken. For LTL sytems this should be 480.
./Markings/SerialNumber/@Visible	Specifies whether to show the serial number on the low resolution image.

./Markings/DateTimeStamp/@Visible	Specifies whether to show the date/time stamp on the low resolution image.
./Markings/ScanId/@Visible	Specifies whether to show the scan id on the low resolution image.
./Markings/Dimensions/@Visible	Specifies whether to show the dimensions on the low resolution image.
./Markings/Indicators/@Visible	Specifies whether to show the indicators (over size, under size, unrefined) on the low resolution image.
./Markings/ItemOutline/@Visible	Specifies whether to show the item outline (bounding box) on the low resolution image.
/CaptureDefinitionDetail/HighResImages/HighResCamCapture	High resolution image parameters. Not applicable to LTL systems.

The following table describes the valid values for **nodim_options**.

No dim option	**Meaning**
None	Dimension all items. This is the default.
Flat	Do not try dimensioning flats – i.e. items that are taller than the flat threshold.
Parcel	Do not try dimensioning parcels- i.e. items than that are not taller than the flat threshold.
All	Do not try dimensioning any items.

The following table describes the valid values for **bctype_list**. Multiple values must be separated by a space.

BC Types	**Meaning**
EAN13	Enable decoding of EAN13 barcodes.
CODE128	Enable decoding of CODE128 barcodes.

CODE39	Enable decoding of CODE39 barcodes.
CODE93	Enable decoding of CODE93 barcodes.
EAN8	Enable decoding of EAN8 barcodes.
UPCE	Enable decoding of UPCE barcodes.
UPCX	Enable decoding of UPCX barcodes.
INT25	Enable decoding of INT25 barcodes.
CODABAR	Enable decoding of CODABAR barcodes.
PATCHCODE	Enable decoding of PATCHCODE barcodes.
DATAMATRIX	Enable decoding of DATAMATRIX barcodes.
QR	Enable decoding of QR barcodes.
PDF417	Enable decoding of PDF417 barcodes.

5.8 System Date and time

5.8.1 Getting system time

Gets system time and date in specified format.

Method signature:

```
QVSystemTime GetSystemTime(string format)
```

format - Format should be +sequence as define here: http://linux.die.net/man/1/date. If no format is provided (i.e. parameter not sent in POST data or specified as empty string), then +%Y%m%d%H%M%S %z %Z will be used, which means YYYYMMDDHHMMSS -0500 EST

5.8.2 Setting system time and time zone

Sets system time and date in specified format.

Method signature:

```
QVServiceResponse SetSystemTime(string time, string timezone)
```

time - Time must use format YYYYMMDDHHMMSS or (e.g. you might use `DateTime.Now.ToString("yyyyMMddHHmmss")` to form the string you later send in C#).

timezone - Time zones are based on Ubuntu Trusty timezones, e.g. "America/New_York", or "US/Eastern". For a complete list, please refer to http://manpages.ubuntu.com/manpages/trusty/man3/DateTime::TimeZone::Catalog.3pm.html

6 Appendix A – Short format URLs

Many API calls can be accessed using a short form URL. The use of these short forms are not recommended in a production system as they can change without notice.

Short form URL (case sensitive) e.g. http://{device}/status	**Production URL (case indifferent)** e.g. http://{device}/WebServices/QubeVuService/Status
/status	/Status
Querying real-time device information	
/status	
Managing tracker engine lifetime	
/restart	
/stop	
/start	
Troubleshooting and maintenance	
/log	
/log_1	
/extractlog	
Querying configuration and device info	
/config	
/options	
/deviceinfo	

Display pages (short URLs *are* intended for production)	
Short form URL (case sensitive)	**Alternate URL**
/display /operatordisplay	/certified/displays/dsdisplay.php (weight panel is shown depending on SuppressScaleData setting)
/customerdisplay	/certified/displays/dsdisplay.php?pgtype=customer (weight panel is shown depending on SuppressScaleData setting)
/dimonlydisplay	/certified/displays/dsdisplay.php?pgtype=dimonly /certified/displays/dsdisplay.php?pgtype=dimonlyoperator
/dimandweightdisplay	/certified/displays/dsdisplay.php?pgtype=dimandweight /certified/displays/dsdisplay.php?pgtype=dimandweightoperator

/dimonlycustomerdisplay	/certified/displays/dsdisplay.php?pgtype=dimonlycustomer
/dimonlyoperatordisplay	/certified/displays/dsdisplay.php?pgtype=dimonlyoperator
/dimandweightcustomerdisplay	/certified/displays/dsdisplay.php?pgtype=dimandweightcustomer
/dimandweightoperatordisplay	/certified/displays/dsdisplay.php?pgtype=dimandweightoperator

7 Appendix B – Sample .Net client code

This appendix contains the source code for a C# class the uses the API defined above for testing purposes. Sample code in other languages such as Java, JavaScript are also available upon request.

1) Create new project
   - Create new Console Application project and name it Dim101_cs
2) Add references
   - Add System.EnterpriseServices reference
   - Add System.Web.Services reference
3) Add web service client proxy
   - Add QubeVuServiceHttpPostClient.cs to project
4) Add sample code
   - Paste following code into Program.cs

```
/*
 * QubeVu Dimensioning Sample Code #1
 *
 * Include QubeVuServiceHttpPostClient.cs in the project.
 *
 * The following references are required by QubeVuServiceHttpPostClient.cs:
 *  System.EnterpriseServices
 *  System.Web.Services
 *  System.Xml
 *
 */
using System;

using QubeVuWebService;                    // needed for QubeVuService class -
 defined in QubeVuServiceHttpPostClient.cs

namespace Dim101_cs
{
    class Program
    {
        static void Main(string[] args)
        {
            // * modify this line below * - replace default IP address of QubeVu with your own unit's IP
            string qubeVuHost = "169.254.1.1";

            QubeVuService qubeVuService;

            //
            // construct web service url using specified hostname
            //
            if (args.Length > 0) { qubeVuHost = args[0]; }
            string webServiceUrl = "http://" + qubeVuHost + "/WebServices/QubeVuService";
            Console.WriteLine("Connecting to {0} ...", webServiceUrl);

            //
            // create an instance of the QubeVu service
            //
            qubeVuService = new QubeVuService();
```

```
            //
            // set web service parameters: url, timeout
            //
            qubeVuService.Url = webServiceUrl;
            qubeVuService.Timeout = 1000;

            //
            // request status from QubeVu
            //
            QVStatus qvStatus = qubeVuService.Status();
            Console.WriteLine("Connection successful");

            //
            // show status
            //
            Console.WriteLine("Status: {0} Capture Id: {1} Ex status: {2}",
                qvStatus.Status, qvStatus.CaptureId, qvStatus.ExtendedStatus);

            // show dims and tracker image url if available
            if (qvStatus.Status == "IMAGING" || qvStatus.Status == "REMOVE")
            {
                Console.WriteLine("Dimensions: {0} x {1} x {2} ({3})", qvStatus.CapturedData.Dimensions.Length,
                                                                       qvStatus.CapturedData.Dimensions.Width,
                                                                       qvStatus.CapturedData.Dimensions.Height
,
                                                                       qvStatus.CapturedData.Dimensions.DimUni
t);
                Console.WriteLine("Tracker image: {0}", qvStatus.CapturedData.TrackerImage.Url);
            }

            // show image url from high resolution camera if available
            if (qvStatus.Status == "REMOVE")
            {
                if (qvStatus.CapturedData.HighResImages != null && qvStatus.CapturedData.HighResImages.Length >
 0)
                {
                    Console.WriteLine("High res image: {0}", qvStatus.CapturedData.HighResImages[0].Url);
                }
            }

            Console.WriteLine("\n\nPress any key to terminate");
            Console.ReadKey(true);
        }
    }
}
```

8 Appendix C – C# .Net code samples

Sample projects were created using Visual Studio 2010 and they target .NET Framework 4 Client Profile. These samples should work fine with lower versions of Visual Studio and .NET Framework as well.

Samples take advantage of the QubeVuService proxy class. The source code for this class is provided, but it can also be generated automatically based on the WSDL file for QubeVuService (in SDK_root\wsdl\QubeVuService.wsdl) using the wsdl.exe tool. The wsdl.exe tool is part of most recent versions of Visual Studio installations.

Most sample projects assume that the device is running at 169.254.1.1 IP address.

8.1 Dim101

Basic demo. Connects to QubeVu, polls for status once, displays dimensions, image URLs. (Same as Appendix A.)

8.2 Dim102

Expands on the previous project by adding extended status monitoring, reports capture id, and capture definition name. Infinite loop polls for status updates continuously.

8.3 Dim103

Expands on the previous project by adding barcode recognition. This sample also has added functionality for saving of all of the low resolution images, also, saving the point coordinates for faces in wireframe model of the scanned object.

8.4 Dim104

Sample demonstrates how to extract and save a rotated and cropped high resolution image of the scanned object. It retains barcode recognition functionality from the previous sample, but does not save low resolution images or wireframe data.

8.5 ExternalTrigger

Demonstrates how to trigger a scan via the "Capture" API call. Also shows how to create a capture definition which enables barcode recognition on the device itself (no need for barcode recognition library on client side).

You may run Dim102 and/or Dim103 while running ExternalTrigger at the same time. When you press any key to trigger a scan/capture in ExternalTrigger you should observe the other windows reporting the status of the device accordingly.

8.6 Barcodes

Demonstrates the use of the QubeVuBarcodesAPI.dll 1D barcode recognition library.

8.7 Barcodes2D

Demonstrates the use of the Postea.QubeVu2DBCRecognition.dll 2D barcode recognition library.

8.8 ImageConversion

Expands on Dim102 sample project. Retrieves dimensions, high resolution image and saves it in both its original format (optional) as well as a rasterized black and white TIFF image for reduced size.

9 Appendix D – Universal Windows Platform Sample (C#)

A UWP sample project is included in the SDK. This sample project has been tested with the latest available Windows IoT OS (at the time of this writing) on Raspberry Pi 3 hardware. Development of UWP apps require the latest Visual Studio 2015 (Update 2 at the time of this writing) as well as the Windows 10 SDK.

The sample takes advantage of the QubeVuService proxy class (QubeVuServiceAsyncClient.cs). This is a different proxy class than what is discussed in the previous section. The asynchronous C# proxy class is available for use with Windows Store apps and Universal Windows Platform development projects (including, but not limited to Windows IoT projects). This class uses the newer Windows.Web.Http namespace in favour of the older System.Net.Http namespace, while also providing only awaitable methods for API calls. This class must be used in conjunction with the code file containing all the QubeVuService type definitions (QubeVuServiceTypes.cs) which has been automatically generated from the WSDL file. Both files can be found in SDK_root\common\src folder.

The UWP sample folder also contains compiled app packages for Intel and ARM architectures as well as instructions on how to sideload the app on Windows 10 desktop/table editions. The README file has instructions on how to deploy the compiled ARM app package on Raspberry Pi 2/3.

9.1 Dim101

Basic demo. Connects to QubeVu, polls for status, displays dimensions.

10 Appendix E – QubeVuBarcodes Client API

The barcode library is implemented as a Windows DLL QubeVuBarcodesAPI.dll (Linux version is available on request) and exposes the following three functions: CreateReader, DestroyReader and RecognizeBarcode. Function signatures and necessary supporting structures are defined in QubeVuBarcodesAPI.cs file in the SDK.

QubeVuBarcodesAPI.dll depends on libiomp5md.dll which is located in the <SDK root>\common\bin folder. Any client application consuming QubeVuBarcodesAPI.dll must have this dependent library present in the same folder where the executable resides.

10.1 Function Reference

CreateReader

Creates a new instance of the barcode reader engine and returns a handle to it.

DestroyReader

Destroys the specified reader and frees any memory associate with it.

RecognizeBarcode

Scans the specified image for barcodes.

11 Appendix D – Postea.QubeVu2DBCRecognition Client API

The 2D barcode library is implemented as a Windows DLL Postea.QubeVu2DBCRecognition.dll (Linux version is available on request) and exposes the BarcodeRecognizer2D class and its ReadImage function. Detailed function signatures and necessary supporting structures are exposed through class metadata which can be viewed in Visual Studio IDE.

11.1 Function Reference

BarcodeRecognizer2D(bcTypes2D)

Creates a new instance of the 2D barcode reader engine and returns a handle to it. bcTypes2D is a bit flag enumeration type corresponding to the available types of 2D barcodes the library supports. Then the barcode type parameter is omitted all available barcode types will be recognized.

ReadImage(Image)

Recognizes 2D barcodes in the image passed to the function. Returns a collection of barcode recognitions.

12 Appendix E – Supported barcodes

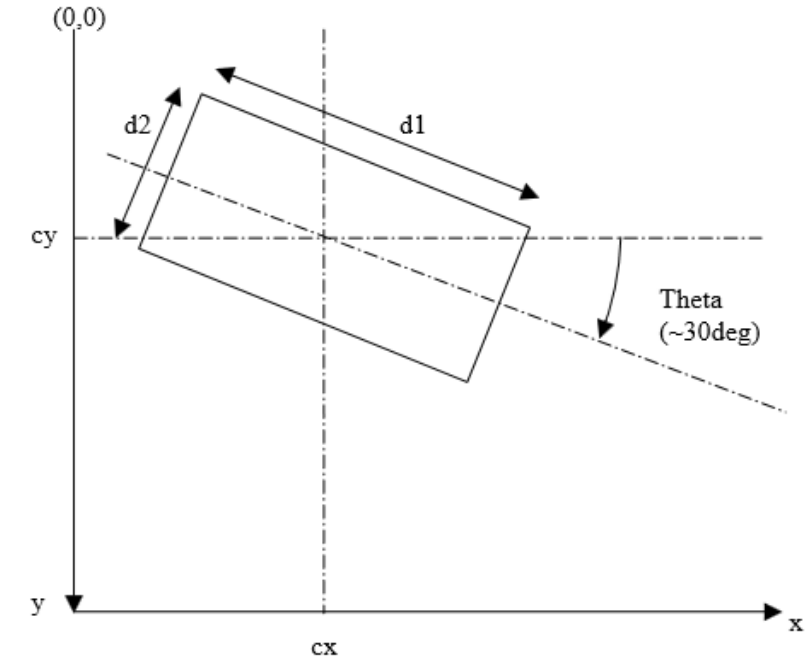
The following barcode types are supported by QubeVu:

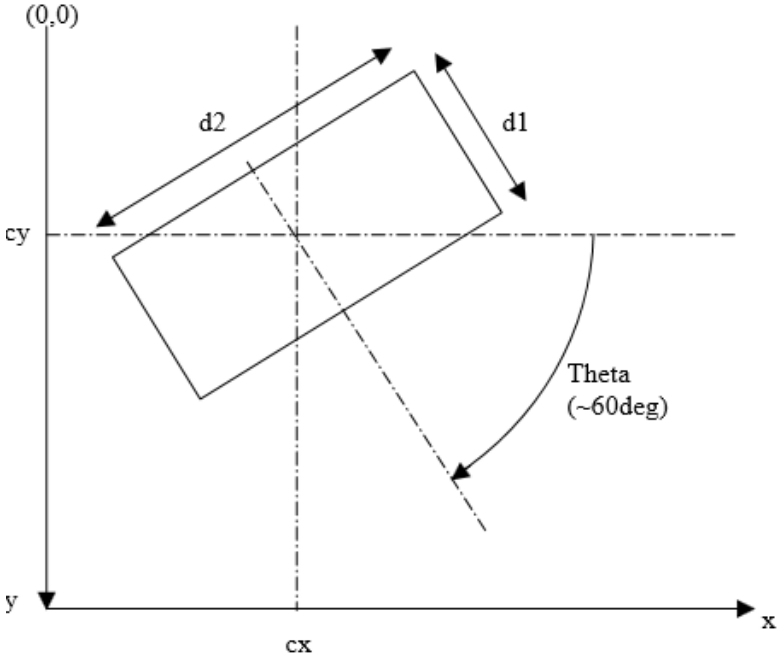
`EAN13, CODE128, CODE39, CODE93, EAN8, UPCE, UPCX, INT25, CODABAR, PATCHCODE`

The following 2D barcode types are supported:

`DATAMATRIX, QR, PDF417`

13 Appendix F – ItemRect illustration





RICE LAKE®
WEIGHING SYSTEMS

230 W. Coleman St. • Rice Lake, WI 54868 • USA
U.S. 800-472-6703 • Canada/Mexico 800-321-6703 • International 715-234-9171 • Europe +31 (0)26 472 1319

www.ricelake.com

January 12, 2017 PN 167741 Rev A