## Contents

1. **Introduction** .......................................................................................................................... 1
   1.1 Overview .................................................................................................................................. 1

2. **Interface Specifications** ......................................................................................................... 2
   2.1 Retrieving Dimensions ............................................................................................................. 2
       2.1.1 Error Codes and Message: ................................................................................................. 6
   2.2 Retrieving the Image .................................................................................................................. 6
   2.3 Retrieving Bar Codes ................................................................................................................. 6
   2.4 Enabling Bar Code Recognition through Capture Definition .................................................. 7
   2.5 Setting the Weight from an External Scale ............................................................................... 7
   2.6 Returned Values ......................................................................................................................... 9
       2.6.1 Status ................................................................................................................................. 9
       2.6.2 Extended Status ................................................................................................................. 10
   2.7 Sample .Net Client Code ......................................................................................................... 11

3. **Appendix** ............................................................................................................................... 16
   3.1 iDimension Bar Codes Client API ............................................................................................ 16
       3.1.1 Reference ......................................................................................................................... 16
   3.2 Supported Bar Codes .............................................................................................................. 18
   3.3 ItemRect Illustration .............................................................................................................. 19

NOTES ............................................................................................................................................. 20

Technical training seminars are available through Rice Lake Weighing Systems. Course descriptions and dates can be viewed at www.ricelake.com/training or obtained by calling 715-234-9171 and asking for the training department.
1.0 Introduction

This document specifies how *iDimension* can be connected to a user’s client system and how that system can then obtain information about an item’s dimensions and the item’s image. It describes only the web service interface to *iDimension*. *iDimension* provides a quick and reliable way to measure 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 bar code recognition and OCR or ICR.

*iDimension* often operates in conjunction with a weigh scale that is collecting the weight of the item. *iDimension* is positioned centrally over the weigh scale so that the dimensions can be captured at the same time as the weight. *iDimension* is able to automatically recognize that an item has been placed underneath it and begin the process of measurement immediately. This information can be available by the time the user’s client system requires it.

During installation, the system is calibrated to suit the location and the specific characteristics of the individual device. Configuration data is held in files on its embedded data storage.

1.1 Overview

*iDimension* connects to the user’s workstation with a single Ethernet connector and requires one power socket.

No specialized 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 an *iDimension* system.

Client processing starts with a user responding to the package being presented by the customer and after the item is placed under the device.

There is a separation between the client support interface and the internal operation of the software. The client interface reads the information currently available and reports 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.

Therefore, the client application will most likely want to implement some form of delayed loop to read the *iDimension* information by monitoring the status on the interface it is calling. The client application 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/ScaleService.\n
The web service supports the HTTP Post binding. The 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.\n
2.0 Interface Specifications

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

Information is only available while an item is on *iDimension* and 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.

2.1 Retrieving 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 is 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 *iDimension*. 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.

```xml
<QVStatus
   CaptureId="string"
   Status="string"
   ExtendedStatus="string"
   OutOfBounds="int"
>
   <Error Code="int" Message="string"/>
   <CapturedData CaptureDefinitionName="string"/>
   <DateTime>string</DateTime>
   <Weight>float</Weight>
   <ScaleData>
     <Weight>int</Weight>
     <ScaleFactor>int</ScaleFactor>
     <IsStable>boolean</IsStable>
     <WeightUnit>string</WeightUnit>
     <RawData>string</RawData>
   </ScaleData>
   <Dimensions>
     <Irregular>boolean</Irregular>
     <Undersize>int</Undersize>
     <Oversize>int</Oversize>
     <Refinement>int</Refinement>
     <DimUnit>string</DimUnit>
     <OutOfBounds>int</OutOfBounds>
     <Height>decimal</Height>
     <Length>decimal</Length>
     <Width>decimal</Width>
   </Dimensions>
</QVStatus>
```

<TrackerImage />
<LowResImages>
   <LowResImage/>
</LowResImages>
<HighResImages>
   <HighResImage/>
   <Barcodes/>
</HighResImages>
</HighResImages>
This is not a blocking call and may require the client to use a delayed loop to call it until it returns the required response.

<table>
<thead>
<tr>
<th>QVStatus</th>
<th>Top level wrapper element</th>
</tr>
</thead>
<tbody>
<tr>
<td>./CaptureId</td>
<td>Sequential capture identifier. CaptureId is incremented each time iDimension processes an item and is reset when the system is restarted.</td>
</tr>
<tr>
<td>./Status</td>
<td>The current status of the device/operation. See below for more details.</td>
</tr>
<tr>
<td>./ExtendedStatus</td>
<td>Extended information about the status of the device/operation. See below for more details.</td>
</tr>
<tr>
<td>./OutOfBounds</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QVStatus/Error</th>
<th>Error details</th>
</tr>
</thead>
<tbody>
<tr>
<td>./Code</td>
<td>Error Code</td>
</tr>
<tr>
<td>./Message</td>
<td>Error Message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QVStatus/CapturedData</th>
<th>Data captured by iDimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>./CaptureDefinitionName</td>
<td>Name of capture definition that has triggered the capture.</td>
</tr>
<tr>
<td>./DateTime</td>
<td>The date and time of the scan.</td>
</tr>
<tr>
<td>./ScaleData</td>
<td>Data reported from attached scale or received through Scale Service interface.</td>
</tr>
<tr>
<td>./Weight</td>
<td>The weight of the item as an integer. Use a scale factor to find the number of decimal places.</td>
</tr>
<tr>
<td>./ScaleFactor</td>
<td>Indicates the number of decimals in ./Weight</td>
</tr>
<tr>
<td>./IsStable</td>
<td>Indicates whether the weight is stable or not.</td>
</tr>
<tr>
<td>./WeightUnit</td>
<td>Specifies the unit in use for the weight.</td>
</tr>
<tr>
<td>./RawData</td>
<td>Hex encoded raw data as received from the scale.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QVStatus/CapturedData/Dimensions</th>
<th>The dimensions of the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>./Irregular</td>
<td>Irregular shaped object (true of false).</td>
</tr>
<tr>
<td>./Undersize</td>
<td>Flag indicating undersize Height (4) or Width (2) or Length (1) or a combination of these.</td>
</tr>
<tr>
<td>./Oversize</td>
<td>Flag indicating oversize Height (4) or Width (2) or Length (1) or a combination of these.</td>
</tr>
<tr>
<td>./Refinement</td>
<td>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)</td>
</tr>
<tr>
<td>./DimUnit</td>
<td>The unit of measure. Valid values are in, mm, cm and m.</td>
</tr>
</tbody>
</table>

Refinement is not required for LTL system.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDimension</td>
<td>Only present if iDimension is declared.</td>
</tr>
<tr>
<td>/outOfBounds</td>
<td>Flag indicating that an item extends out of the sensor's field.</td>
</tr>
<tr>
<td>/Length</td>
<td>Length of object (longer dimension in the X,Y plane).</td>
</tr>
<tr>
<td>/Width</td>
<td>Width of object (shorter dimension in the X,Y plane).</td>
</tr>
<tr>
<td>/Height</td>
<td>Height of object (dimension in the Z plane).</td>
</tr>
<tr>
<td>/QVStatus/CapturedData/RawDimensions</td>
<td>Included if raw (unrounded) dimensions are enabled</td>
</tr>
<tr>
<td>/Length</td>
<td>Length of object (longer dimensions in the X,Y plane)</td>
</tr>
<tr>
<td>/Width</td>
<td>Width of object (shorter dimension in the X,Y plane)</td>
</tr>
<tr>
<td>/Height</td>
<td>Height of object (dimension in the Z plane)</td>
</tr>
<tr>
<td>/QVStatus/CapturedData/LowResImages</td>
<td>Wrapper for zero or more LowResImage elements</td>
</tr>
<tr>
<td>/@Url</td>
<td>URL of low-res image.</td>
</tr>
<tr>
<td>/@Name</td>
<td>URL of low-res image.</td>
</tr>
<tr>
<td>/ItemRect</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/CenterX</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/CenterY</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/D1</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/D2</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/Theta</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemWireframe</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>/ItemWireframe/Faces</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>/ItemWireframe/Faces/Face</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>/Vertices</td>
<td>Multiple points may be defined under this element.</td>
</tr>
<tr>
<td>/Point</td>
<td>X pixel coordinate of the vertex.</td>
</tr>
<tr>
<td>/X</td>
<td>X pixel coordinate of the vertex.</td>
</tr>
<tr>
<td>/Y</td>
<td>Y pixel coordinate of the vertex.</td>
</tr>
<tr>
<td>/QVStatus/CapturedData/HighResImages</td>
<td>Wrapper for zero or more HighResImage elements</td>
</tr>
<tr>
<td>/@Url</td>
<td>URL of high-res image.</td>
</tr>
<tr>
<td>/@Name</td>
<td>URL of high-res image.</td>
</tr>
<tr>
<td>/ItemRect</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/CenterX</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/CenterY</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/D1</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/D2</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemRect/Theta</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>/ItemWireframe</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>/ItemWireframe/Faces</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>/ItemWireframe/Faces/Face</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>/Vertices</td>
<td>Multiple points may be defined under this element.</td>
</tr>
<tr>
<td>/Point</td>
<td>X pixel coordinate of the vertex.</td>
</tr>
<tr>
<td>/X</td>
<td>X pixel coordinate of the vertex.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Y</td>
<td>Y pixel coordinate of the vertex.</td>
</tr>
<tr>
<td>.//Barcodes</td>
<td>Multiple bar code elements, one for each bar code found, may appear under this item.</td>
</tr>
<tr>
<td>.//Barcodes/Barcode</td>
<td>Details of a single bar code.</td>
</tr>
<tr>
<td>.//RawData</td>
<td>Bar code value.</td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage</td>
<td>Details of low resolution image used for finding item</td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/@Url</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/@Name</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemRect</td>
<td>Either this element or ItemWireframe is filled in.</td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemRect/CenterX</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemRect/CenterY</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemRect/D1</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemRect/D2</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemRect/Theta</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe</td>
<td>Either this element or ItemWireRect is filled in.</td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe/Faces</td>
<td>Multiple faces may be defined.</td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe/Faces/Face</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe/Faces/Face/Vertices</td>
<td>Multiple points may be defined under this element.</td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe/Faces/Face/Vertices/Point</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe/Faces/Face/Vertices/Point/X</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/ItemWireframe/Faces/Face/Vertices/Point/Y</td>
<td></td>
</tr>
<tr>
<td>.//QVStatus/CapturedData/TrackerImage/QVStatus/Crc</td>
<td>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.</td>
</tr>
</tbody>
</table>
2.1.1 Error Codes and Message:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Hardware Initialization FAILED.</td>
</tr>
<tr>
<td>2</td>
<td>Tracker Config Initialization FAILED.</td>
</tr>
<tr>
<td>3</td>
<td>Missing RegistrationMarksCropped.bmp.</td>
</tr>
<tr>
<td>4</td>
<td>Setting reference image for Targetfinder FAILED.</td>
</tr>
<tr>
<td>5</td>
<td>Loading of Calibration files FAILED.</td>
</tr>
<tr>
<td>6</td>
<td>Getting new Images from hardware FAILED.</td>
</tr>
<tr>
<td>7</td>
<td>Tracking FAILED.</td>
</tr>
<tr>
<td>8</td>
<td>Calibrating.</td>
</tr>
<tr>
<td>9</td>
<td>TCP Server Port binding failed</td>
</tr>
<tr>
<td>10</td>
<td>TCP Server exception in Processing Client</td>
</tr>
<tr>
<td>11</td>
<td>TCP Server time out on Imaging</td>
</tr>
<tr>
<td>12</td>
<td>Low resolution camera needs to be calibrated first!</td>
</tr>
<tr>
<td>13</td>
<td>Calibration stopped.</td>
</tr>
<tr>
<td>14</td>
<td>Error loading/parsing Configuration.</td>
</tr>
<tr>
<td>15</td>
<td>Unable to save Calibration to file.</td>
</tr>
<tr>
<td>16</td>
<td>Unable to use name set in Capture/Get Command. CaptureDefinition with name were not set.</td>
</tr>
<tr>
<td>17</td>
<td>Invalid CaptureDefinition command.</td>
</tr>
<tr>
<td>18</td>
<td>Unable to delete Calibration files.</td>
</tr>
<tr>
<td>19</td>
<td>Unable to Zero Height.</td>
</tr>
</tbody>
</table>

Table 2-1. Error Codes and Messages

2.2 Retrieving the Image
While designing the client dialogue that uses iDimension, it should be kept in mind that the iDimension 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 of the images to the workstation also takes a finite time.

The image returned is of the top face of the 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 color depth and resolution of the image depends on the parameters specified in the capture definition.

2.3 Retrieving Bar Codes
Reading bar codes requires the capture of an image by the high resolution camera typically at 140dpi or better, but the actual resolution will depend on the bar code to be recognized. iDimension provides two approaches to retrieve bar codes from the captured high resolution images.

One approach is to simply specify the types of bar codes of interest in the capture definition. Once this is set up, the device will search for bar codes in the corresponding high resolution images and will return the values of each bar code found in response to a status along with the high resolution images’ URLs. The found bar codes are returned in the bar codes element of each HighResImage.

Another approach is to use the QubeVuBarcodes Client API. This API can be used by the client application to find bar codes from a high resolution image that has been captured. The API is simple to use and is described in detail in Section 3.1 on page 16.
The benefit of using the client API is that it can take advantage of the fast processors of the client hardware to receive results faster.

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

### 2.4 Enabling Bar Code Recognition through Capture Definition

Use the create capture definition interface: `QubeVuService/CreateCaptureDefinition`

**Method Signature:**

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

- `name` - The name of the capture definition to be created
- `definitionString` - The XML string describing the capture definition.

For example:

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

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

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

### 2.5 Setting the Weight from an External Scale.

In a typical configuration, a weighing scale is connected to *iDimension* 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 device. One such scenario is where there is an existing client application that requires direct control of the scale. Operating *iDimension* 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 the system of any weight changes.

It is important that the weight change notification happens quickly to ensure that it is in sync with what the scan head sees.
The client application must use iDimension'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 iDimension of any weight changes, regardless of whether the weight is zero or stable.

It is important to note that when iDimension 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 iDimension, but must be communicated 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 or 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.

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

2.6 Returned Values

2.6.1 Status

<table>
<thead>
<tr>
<th>Status value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTING</td>
<td>The service is starting up.</td>
</tr>
<tr>
<td>STARTED</td>
<td>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 it, 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.</td>
</tr>
<tr>
<td>READY</td>
<td>The device is ready and waiting to be used – there is no item on it.</td>
</tr>
<tr>
<td>TRACKING</td>
<td>The device is processing a change in image after an item has been placed under it.</td>
</tr>
<tr>
<td>IMAGING</td>
<td>The device measurements have been determined and the camera is being adjusted to take the image.</td>
</tr>
<tr>
<td>REMOVE</td>
<td>The image has been fully processed – the item can be removed when the client processing has completed.</td>
</tr>
<tr>
<td>WAIT</td>
<td>Preparing the device for the next item. The previous image and dimensions are deleted from memory and the lens is reset.</td>
</tr>
<tr>
<td>STOPPED</td>
<td>The service has stopped – there is a problem.</td>
</tr>
<tr>
<td>CALIBRATING</td>
<td>The device is in calibration mode.</td>
</tr>
<tr>
<td>CONFIGURING</td>
<td>The device is in configuration mode.</td>
</tr>
</tbody>
</table>

Table 2-2. Status Values
### 2.6.2 Extended Status

The returned extended status attribute contains zero or more comma separated strings that provide additional information about the status of the device.

<table>
<thead>
<tr>
<th>Constraint value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScaleNotStable</td>
<td>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.</td>
</tr>
<tr>
<td>MotionDetected</td>
<td>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.</td>
</tr>
<tr>
<td>ItemDetected</td>
<td>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.</td>
</tr>
<tr>
<td>ItemNotDetected</td>
<td>This is set when the system is in ready mode and there is no item on the platform/scale.</td>
</tr>
<tr>
<td>TrackerNotConfident</td>
<td>This indicates that the tracker detected an item but it is not confident of the dimensions of the item. After a timeout (configurable) the system will progress to next step and return zero-valued dimensions.</td>
</tr>
<tr>
<td>ExceptionOccurred</td>
<td>This is set when an exception occurs.</td>
</tr>
<tr>
<td>DeviceNotStable</td>
<td>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.</td>
</tr>
</tbody>
</table>

*Table 2-3. Extended Status Values*
2.7 Sample .Net Client Code

This appendix contains the source code for a C# class that 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 Windows Forms Application project and name it QubeVuSampleClient
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 controls to form
   • Add two buttons to Form1 and name them 'status' and 'capture'
   • Add two text boxes to Form1 and name them 'txtResult' and 'txtDims'
5. Add sample code
   • Paste following code into Form1.cs

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using QubeVuWebService;
using System.IO;
using System.Xml.Serialization;
using System.Net;

namespace QubeVuSampleClient
{
    public partial class Form1 : Form
    {
        private QubeVuService qvService;
        private Uri qvServiceUri;

        public Form1()
        {
            InitializeComponent();

            // create form layout
            LayoutForm();

            // register event handlers for the buttons
            this.status.Click += new System.EventHandler(this.status_Click);
            this.capture.Click += new System.EventHandler(this.capture_Click);

            // create instance of service and set url - change host name according to setup
            qvService = new QubeVuService();
            qvService.Url = "http://QubeVu/WebServices/QubeVuService";
            qvServiceUri = new Uri(qvService.Url);
        }
    }
}
```
// create a capture definition to be used when the capture button is pressed
// this may be done programmatically or through the QV Manager UI
CreateCaptureDefinition();

private void CreateCaptureDefinition()
{
    // check for existence of capture definition - production app would also update
    // definition when it is inconsistent with desired definition
    // note that this could be managed from a separate app; capture definitions are saved
    // by qv
    CaptureDefinitionDetail captureDef = qvService.GetCaptureDefinition("SampleClient");

    if (captureDef != null)
    {
        ShowResult("GetCaptureDefinition", ResponseToXmlString(captureDef.GetType(),
            captureDef));
        return;
    }

    // create capture definition for external trigger
    string lowresCamCapture = @"<LowResCamCapture ImageName=""LowResImage1"">" +
        @"<ResX>640</ResX>" +
        @"<ResY>480</ResY>" +
        @"</LowResCamCapture>";

    string captureDefinition = @"<CaptureDefinitionDetail Name=""SampleClient"">" +
        @"<NoDimItems>Flat</NoDimItems>" +
        @"<LowResImages>" + lowresCamCapture +
        @"</LowResImages>" +
        @"<HighResImages>" +
        @"</HighResImages>" +
        @"</CaptureDefinitionDetail>";

    QVServiceResponse response = qvService.CreateCaptureDefinition("SampleClient",
        captureDefinition);
    ShowResult("CreateCaptureDefinition", ResponseToXmlString(response.GetType(),
        response));
}

// get current status from QubeVu
private void status_Click(object sender, EventArgs e)
{
    // get current status from server
    QVStatus qvStatus = qvService.Status();
    ShowResult("Status", ResponseToXmlString(qvStatus.GetType(), qvStatus));

    // show status to user and save images if available
    ShowStatus(qvStatus);
}
private void capture_Click(object sender, EventArgs e)
{
    // get last capture id
    QVStatus qvStatus = qvService.Status();
    string lastCaptureId = qvStatus.CaptureId;
    string currentCaptureId = "";

    // initiate capture
    QVServiceResponse response = qvService.Capture("SampleClient");
    ShowResult("Capture("SampleClient")", ResponseToXmlString(response.GetType(), response), "Waiting for results ...");

    // wait for results of capture - this would typically be done from a timer and not inline so the UI is not blocked while waiting
    do
    {
        // poll up-to-date status and get latest capture id
        try
        {
            qvStatus = qvService.Status();
            currentCaptureId = qvStatus.CaptureId;
        }
        catch
        {
            //
        }

        if (string.Compare(lastCaptureId, currentCaptureId, true) != 0)
        {
            if (string.Compare(qvStatus.Status, "REMOVE", true) == 0)
            {
                // if capture id has changed and status is removed then we're done with capture
                break;
            }

            // show intermediate status while waiting
            ShowStatus(qvStatus);
        }
    } while (true);

    // show completed status
    ShowResult("Capture("SampleClient")", ResponseToXmlString(response.GetType(), response), "Capture complete.");
    ShowStatus(qvStatus);
}

private void ShowStatus(QVStatus qvStatus)
{
    // got dimensions?
    if (string.Compare(qvStatus.Status, "IMAGING", true) == 0 || string.Compare(qvStatus.Status, "REMOVE", true) == 0)
{ // show dimensions
    Dimensions dims = qvStatus.CapturedData.Dimensions;
    txtDims.Text = string.Format("Capture [{0}]: {2} x {3} x {4} ({1})", qvStatus.CaptureId, qvStatus.Status, dims.Length, dims.Width, dims.Height);
}

// got images?
if (string.Compare(qvStatus.Status, "REMOVE", true) == 0)
{
    // get image url(s) from response
    string fileName = "TrackerImage.bmp";

    // download image(s)
    WebClient webClient = new WebClient();
    webClient.BaseAddress = qvServiceUri.GetComponents(UriComponents.SchemeAndServer, UriFormat.UriEscaped);

    webClient.DownloadFile(url, fileName);
}

txtDims.Refresh();
}

#region "Helper functions"
private void ShowResult(string method, string response)
{
    ShowResult(method, response, null);
}

private void ShowResult(string method, string response, string ps)
{
    txtDims.Text = "";

    txtResult.Text = string.Format("[{}]", method);
    txtResult.Text += Environment.NewLine + response;
    if (!string.IsNullOrEmpty(ps))
    {
    }

    txtResult.Refresh();
}
private void LayoutForm()
{
    this.status.Location = new System.Drawing.Point(12, 23);
}
this.status.Size = new System.Drawing.Size(75, 23);
this.status.Text = "Status";

this.capture.Location = new System.Drawing.Point(12, 52);
this.capture.Size = new System.Drawing.Size(75, 23);
this.capture.Text = "Capture";

this.txtResult.Location = new System.Drawing.Point(111, 52);
this.txtResult.Size = new System.Drawing.Size(480, 151);
this.txtResult.Multiline = true;

this.txtDims.Location = new System.Drawing.Point(111, 23);
this.txtDims.Size = new System.Drawing.Size(480, 20);

this.ClientSize = new System.Drawing.Size(615, 262);
this.Text = "QubeVu Sample Client";

} // helper function to serialize service responses to xml string
// this is only needed to show the responses - no need in production
internal static string ResponseToXmlString(Type type, object response)
{
    XmlSerializer serializer = new XmlSerializer(type);
    using (StringWriter stringWriter = new StringWriter())
    {
        serializer.Serialize(stringWriter, response);
        return stringWriter.ToString();
    }
} #endregion
3.0 Appendix

3.1 iDimension Bar Codes Client API
The bar code 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.

3.1.1 Reference

Function CreateReader
Creates a new instance of the bar code and returns a handle to it.
Syntax:
void *CreateReader(char *config);

Function DestroyReader
Destroys the specified reader and frees any memory associate with it.
Syntax:
DestroyReader(void *readerHandle);

Function RecognizeBarcode
Scans the specified image for bar codes.
Syntax:
Int RecognizeBarcode(void *readerHandle, unsigned char *hBmp, int bufferLength, SEARCH_AREA *searchAreaParm, int cropped, EymBCReaderResult resultsBuffer[]);

Struct SEARCH_AREA
typedef struct
{
    float left;
    float top;
    float width;
    float height;
} SEARCH_AREA;

Struct EymBCReaderResult
typedef struct {
    char rawData[100];  // complete decoded payload (ASCII) */
    char decodedData[500]; // interpreted payload (ASCII) */
    char symbology[100]; // decoded symbology */
    char codeType[100];  // description of the code type (e.g. Generic; UCC/EAN-128; etc) */
    char description[100]; // description of the data (e.g. <none>, Product ID, dimension, etc. */
    float confidence; // confidence measure (in [0.0 to 1.0]) */
    float redundancy; //
    int imgXa, imgYa; // location of the beginning of decoded barcode within the processed image */
int imgXb, imgYb; // location of the beginning of decoded barcode within the processed image */
int imgXc, imgYc; // location of the barcode hot spot within the processed image */

} EymBCReaderResult;

C# Definitions
private static extern IntPtr CreateBarcodeReader(string config);

private static extern void DestroyBarcodeReader(IntPtr readerHandle);

private static extern int RecognizeBarcode(IntPtr readerHandle, IntPtr bmHandle, int bufLength, ref SEARCH_AREA searchArea, int cropped, ref EymBCReaderResults results);

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
private struct EymBCReaderResults
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 64)]
    public EymBCReaderResult[] results;
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
private struct EymBCReaderResult
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
    public string rawData; // complete decoded payload (ASCII) */
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 500)]
    public string decodedData; // interpreted payload (ASCII) */
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
    public string symbology; // decoded symbology */
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
    public string codeType; // description of the code type (e.g. Generic; UCC/EAN-128; etc) */
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
    public string description; // description of the data (e.g. <none>, Product ID, dimension, etc. */
    public float confidence; // confidence measure (in [0.0 1.0]) */
    public float redundancy; // reports the redundancy measured from the associates super-res BC signal */
    public int imgXa, imgYa; // location of the beginning of decoded barcode within the processed image */
    public int imgXb, imgYb; // location of the beginning of decoded barcode within the processed image */
    public int imgXc, imgYc; // location of the barcode hot spot within the processed image */
}

private struct SEARCH_AREA
{
    public SEARCH_AREA(float x, float y, float width, float height)
    {
        this.left = x;
    }
this.top = y;
this.width = width;
this.height = height;
}

float left;
float top;
float width;
float height;
}

.Net Sample Code

// include C# definitions from above

Image img = QubeVuHighResImage;

// Init barcode reader API – this can be done when app starts
IntPtr barcodeReader = CreateBarcodeReader("");

// convert from Image class to unmanaged memory
byte[] imageBytes = null;
using (MemoryStream ms = new MemoryStream())
{
    img.Save(ms, ImageFormat.Bmp);
    imageBytes = ms.ToArray();
}
IntPtr imageBuffer = Marshal.AllocHGlobal(imageBytes.Length);
Marshal.Copy(imageBytes, 0, imageBuffer, imageBytes.Length);

EymBCReaderResults results = new EymBCReaderResults();
SEARCH_AREA searchArea = new SEARCH_AREA(0, 0, 1, 1);

// find barcodes
RecognizeBarcode(barcodeReader, imageBuffer, imageBytes.Length, ref searchArea, 0, ref results);

// free resources from barcode reader – this can be done when app closes
DestroyBarcodeReader(barcodeReader);

// found barcodes can be enumerated from ‘results’

3.2 Supported Bar Codes
The following bar code types are supported by iDimension:
EAN13, CODE128, CODE39, CODE93, EAN8, UPCE, UPCX, INT25, CODABAR, PATCHCODE
3.3 ItemRect Illustration

Theta (~30 deg)

Theta (~60 deg)