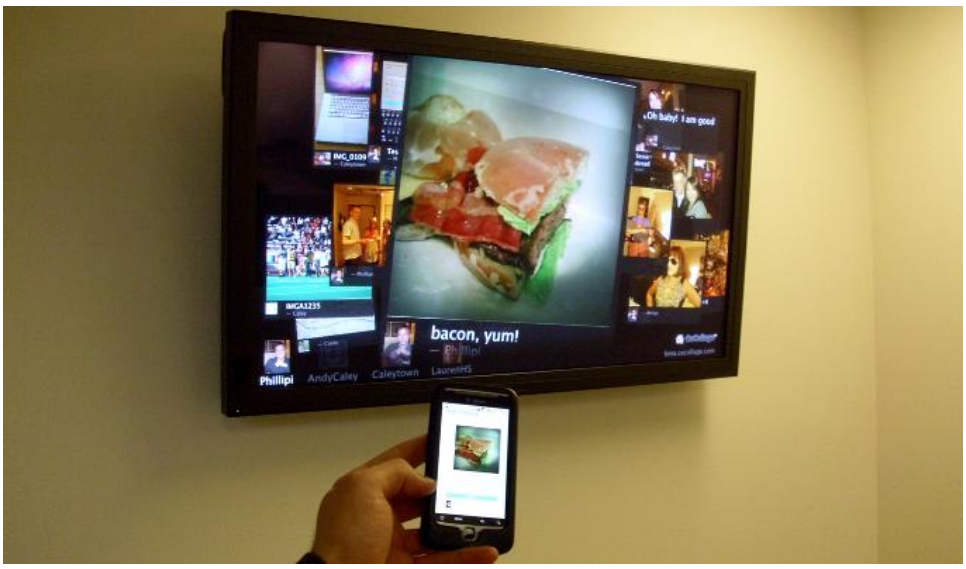
# TwilioCollage

*Render photos collectively as a collage using Twilio and the Windows Developer Preview.*

## Contents

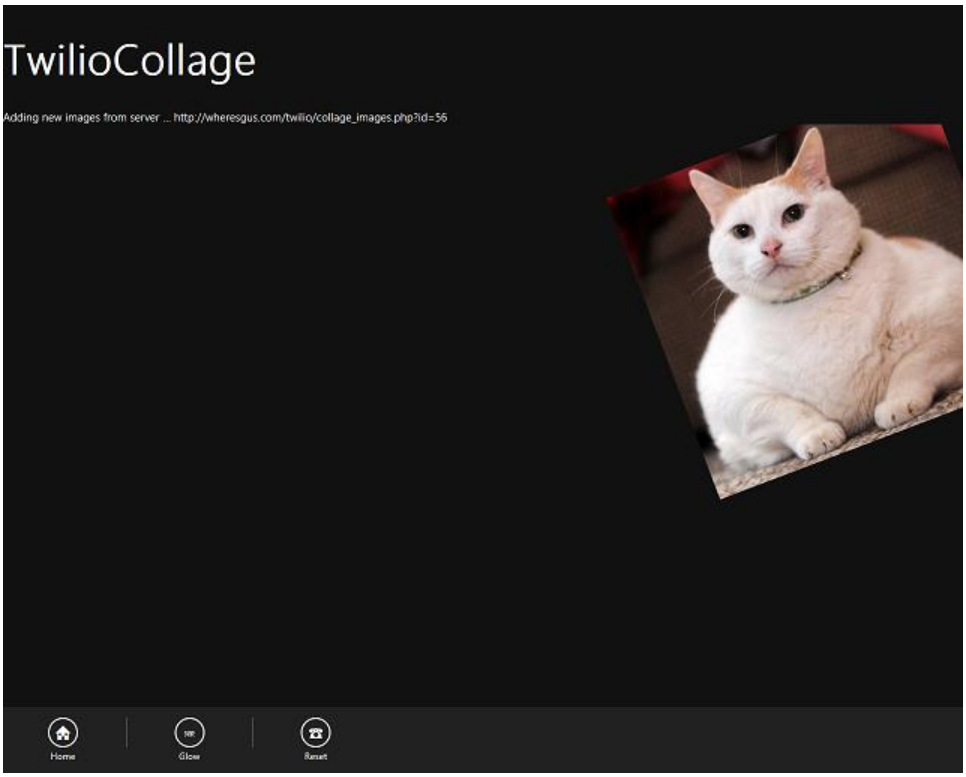
# Introduction

Have you ever seen CoCollage? It's a cool product that lets people share photos with each other onto a TV screen by sending text messages.




I was trying to think of something cool to play with for the Windows Developer Preview and Twilio and I couldn't stop thinking about CoCollage. I mostly just wondered how easy it would be to clone with the new Windows development model and the simple and powerful Twilio API for voice and text services.

In roughly a week or so, I have put together a demo that lets you send images to a service that then renders the images within a Metro-style app. The following image shows the app running:




The way the app works is the user sends a text message to a Twilio number with a URL. After the text message is sent, the URL is validated and then the image is added to the collage rendered in a Metro style app.

## Setting It Up

Set up the MySQL databases for the backend.

Create the images database: “create table images(id int auto_increment primary key not null, insertDate DATETIME, uploadedBy varchar(14), URL varchar(128));”

Create the actions database: “create table actions(id int auto_increment primary key not null, action varchar(32), options varchar(32));”

Update the configuration settings section in collage_images.php, collage_actions.php, handle_call.php, twiliocollage.php, and admin_ivr.php. Once you have set up the files, copy everything from the sources\services folder to your web server.

Point Twilio to your SMS handler from the Twilio User Account page by setting the SMS URL to the web address for twiliocollage.php.

Update the configuration section of the TwilioCollage Windows 8 client by editing the configuration parameters in the default.js file. You could always just use my demo pages by setting svcBaseURL to, http://wheresgus.com/twilio/ to see the content that is currently listed in my service.

## Using TwilioCollage

First, set up all of the variables as described in Setting it Up.

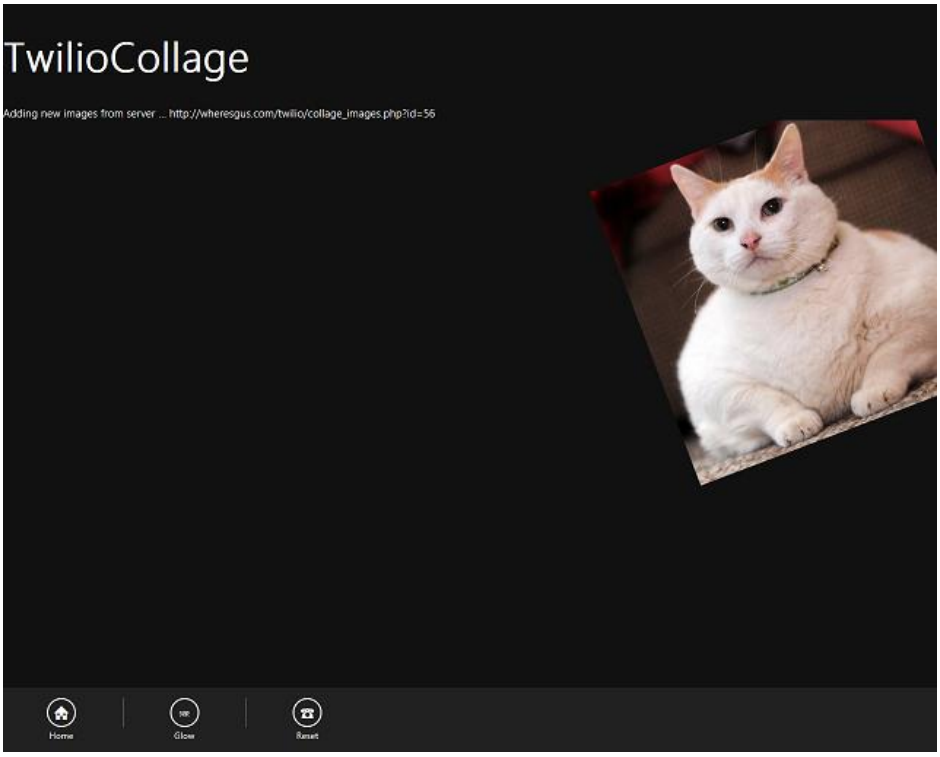
Start the client application, TwilioCollage, on your Windows developer preview PC. You can find the client in the **Client** folder of the sources. After you have started the client application, you will see an empty collage that should look something like this:


To use the actual Collage tool, send a text message from your phone to your Twilio number with a URL in the text message. For example, if you are using the sandbox and your sandbox pin is 1234-4321, send a text message:

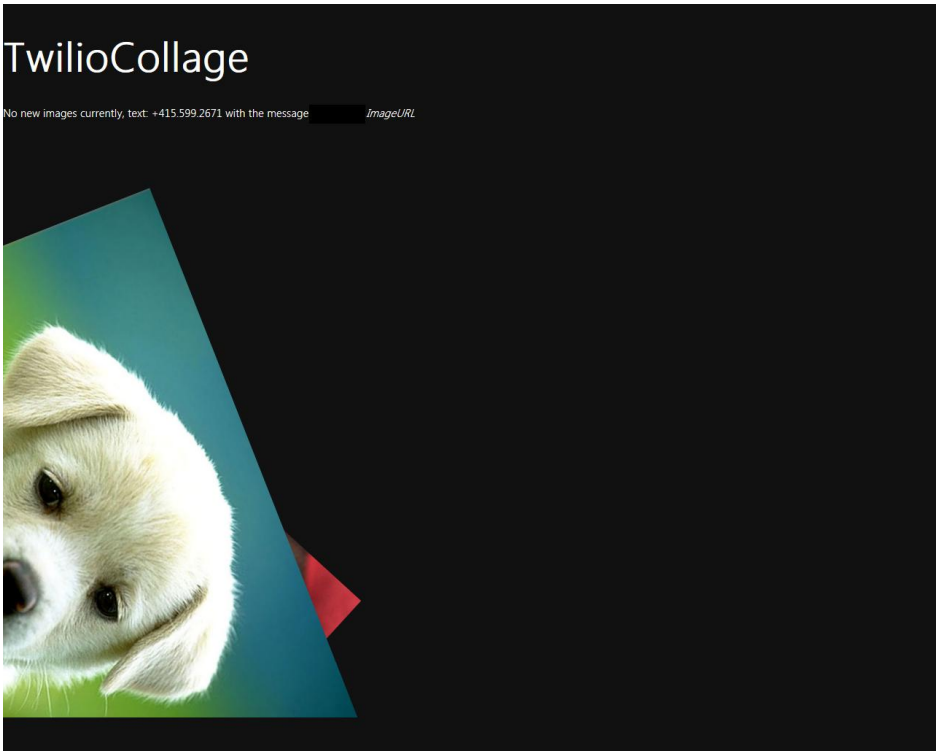
to: 415.599.2671

msg: 1234-4321 http://wheresgus.com/cat.png

After sending that text message, you will see a screen like the following:




You can keep adding images to the collage while it runs. The following screen shows a collage with more than one image:



If you want to reset the entire collage, send a text message to the Twilio number with the sandbox code (as necessary) and the word reset in the body.  Note, currently this will not call any user, you must be whitelisted as an administrator to trigger the reset.

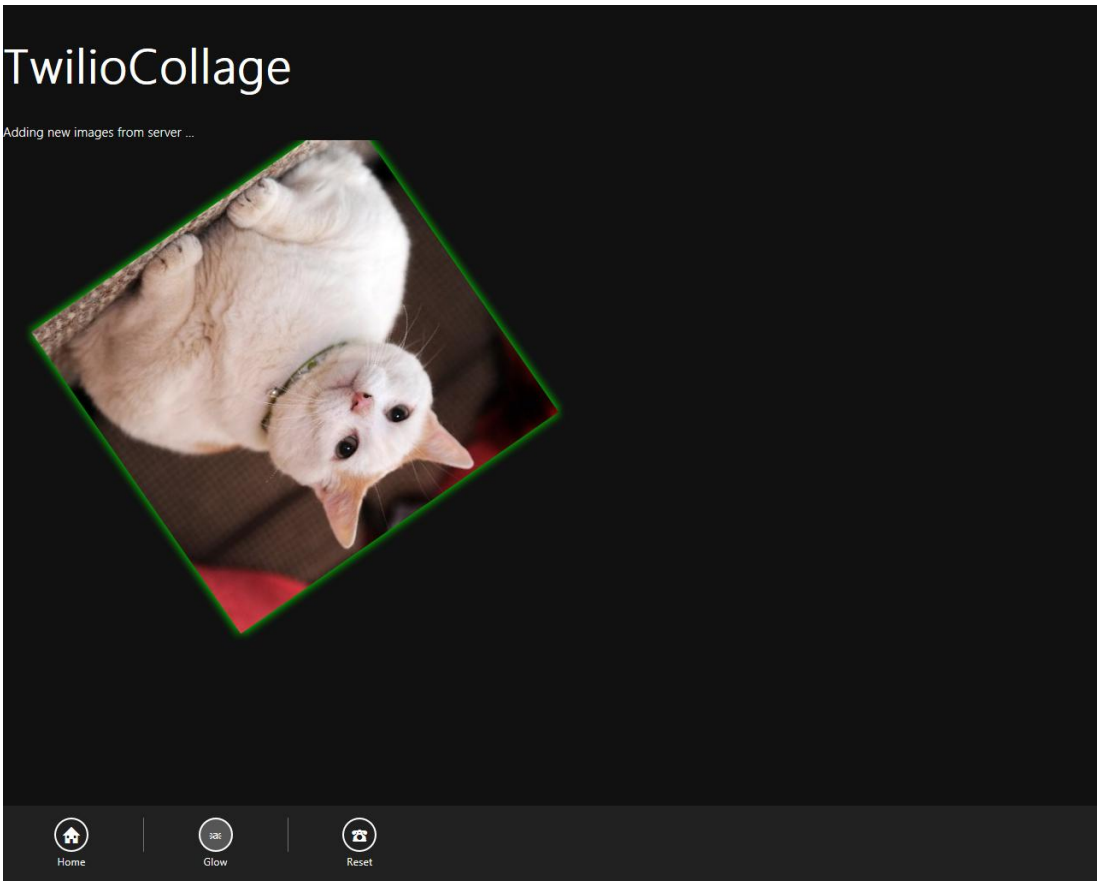
to: 415.599.2671

msg: 1234-4321 reset

When the reset command is triggered, you will receive a call with an IVR menu.  When prompted, press 1 on your phone's keypad and the collage will reset.

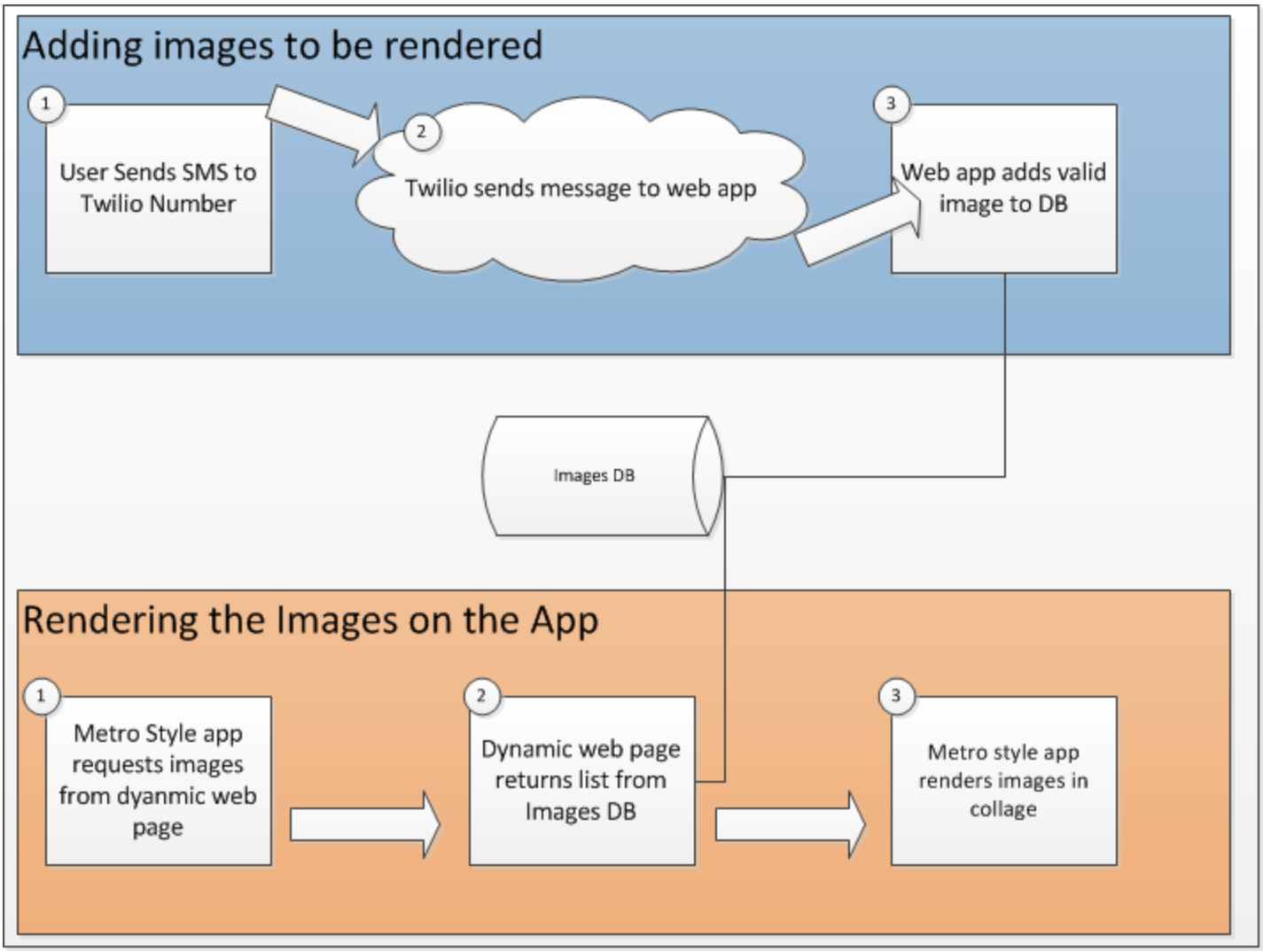
From the client, you can also initiate the IVR reset by right-clicking to bring up the app bar and then clicking the Reset button.

There are various tweaks that are in the Client that you could extend to have effects, the following screenshot shows a glow effect:




## How does it work?

You have a general idea how the application is used at a high level, the following diagram shows how the various pieces fit together in the app and service.




First, a user sends a text message to the Twilio number (1). Twilio parameterizes the text message and sends the SMS object to a web application that can split out parameters sent within the message (2). The web application splits out the URL from the request that it receives from Twilio then either responds with an error code or confirms that the image was valid. The response is sent back to the user using Twilio. If the image was a valid image, the web application adds an entry to its table of images and we're ready to render the image (3).

The TwilioCollage application runs continuously on Windows. Every 5 seconds, the app refreshes its list of images from a dynamic web page (1). This page renders the list as XML by using the WinJS XHR object. The XHR object simplifies getting the updates over the internet. Because each image is assigned a unique and sequential ID when it's added to the image DB, the application knows which images are new and which are already being rendered based on the largest id that it has associated with an image. Rendered images are appended to a hidden DIV that exists on the Metro style app and are added to a list that is rendered within a HTML5 canvas object in the app's HTML (3).

I'll briefly go into more details on the client and server in the next two sections.

## Services: Twilio and the Web Handlers

### Adding an Image

First, the user sends an SMS text to the Twilio number. Twilio then POSTS data to the URL specified in the SMS handler script which is configured on the Twilio user account page which should be set to *<somehost/somefolder>/*twiliocollage.php. When the data comes from Twilio to the page, information sent within the text can be extracted from the page request. For this demo, the from value and message body are used. The handler will author the response as XML. The following code shows how the XML response is started and the 'From' and 'Body' data are extracted from the request:

```
echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
$from = $_REQUEST['From'];
$body = $_REQUEST['Body'];
```

You now will have the caller's number as the *$from* variable and the SMS message body as the *$body* variable. Using some PHP magic, the image is tested to be valid with the getimagesize() command.

```
$size   = getimagesize($body);
$s_ok   = 0;
$width  = -1;
$height = -1;
if ($size) {
   $width = $size[0];
   $height = $size[1];
   $s_ok = 1;
} else {
   // error
   $message = "Error: url wasn't an image";
```

If you have a valid image, you then store the image in the images database.

```
if ($s_ok == 1){
   // Connecting, selecting database
   $link = mysql_connect($host, $user, $pass)
      or die('<response><sms>Could not connect: ' . mysql_error() . '<sms></response>');
   mysql_select_db($db) or die('<response><sms>Could not select database</sms></response>');

// TODO: mysql_real_escape_string(
   mysql_query("insert into images(insertDate, uploadedBy, URL) values (NOW(), '$from', '$body')");

   // Closing connection
   mysql_close($link);
   $message = 'Upload succeeded! Width:' . $width . " Height:" . $height;
```

You now will have added an image to the DB for rendering. The XML for the response is created and then a text message is sent back to the user.

```
<Response>
<Sms><?=$message?></Sms>
</Response>
```

The client will periodically request the collage_images.php script which will list all of the images currently in the DB. The following code shows how this XML is rendered for the client:

```
if ($s_ok == 1){
   // Connecting, selecting database
   $link = mysql_connect($host, $user, $pass)
      or die(mysql_error());
   mysql_select_db($db)

   $query = "select id, insertDate, uploadedBy, URL from images where id > " . $id;
   $resultset = mysql_query($query); // TODO: where id > ?id

   echo "<response>\n";

   while ($row = mysql_fetch_assoc($resultset)) {
      echo "<twImage>\n";
      echo "  <id>" . $row['id'] . "</id>";
      echo "  <insertDate>" . $row['insertDate'] . "</insertDate>\n";
      echo "  <uploadedBy>" . $row['uploadedBy'] . "</uploadedBy>\n";
      echo "  <src>" . $row['URL'] . "</src>\n";
      echo "</twImage>\n";
   }

   echo "</response>\n";
   // Closing connection
   mysql_close($link);
```

## Resetting the Collage

Just as with the adding an image routine, the collage is reset by initiating a text message. The difference is that the message must be from an "administrator" phone number and have the body set to 'reset'. This script uses the same handler as the Adding an Image script (twiliocollage.php) and so in the following example code, we'll already have everything setup for testing against the administrator phone number. The following code shows how the Twilio PHP library is used to initiate a call with the administrative user that then directs Twilio to place a call to the person who texted prompting for a reset:

```
// ADMIN MODE ##
// Can call reset command, etc...
if ($from == $adminphone){
   if ($body == 'reset'){
      $wasCmd  = 'true';
                              // perform reset sequence on #'s
```

```
        // Twilio REST API version
        $version = '2010-04-01';

        // Set account SID and AuthToken
        $client = new Services_Twilio($sid, $token, $version);

        $handlerURL = $baseURL + "handle_call.php";

        try{
              $call = $client->account->calls->create(
                    $phonenumber, $from, $handlerURL
              );
        }catch (Exception $e){
          $message = 'Error: ' . $e->getMessage();
        }
        $message = "Collage reset initiated!";
    }
```

When the client executes the **calls->create** method, Twilio initiates an IVR script based on the response created by handle_call.php. The IVR script in handle_call.php just generates the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Response>
  <Gather numDigits="1" action="http://wheresgus.com/twilio/admin_ivr.php">
    <Say>Thank you for initiating the administrative interface for twilio collage</Say>
    <Say>To reset your collage, press 1.</Say>
  </Gather>
  <Say>Sorry, I didn't get your response.</Say>
  <Redirect>http://wheresgus.com/twilio/handle_call.xml</Redirect>
</Response>
```

Twilio parses the Response XML and then asks the user if they want to reset the collage. If the user gives numeric input, Twilio then runs the admin_ivr.php script. The admin_ivr.php script parses the input from Twilio as shown in the following code:

```
  $user_pushed=(int) $_REQUEST['Digits'];
```

If the input was 1, then the images are deleted from the database and the actions database is updated with a message that will cause the client to re-initialize its images.

```
  if (($user_pushed) == 1){
    echo "    <Say>Triggering collage reset</Say>\n";
    // Connecting, selecting database, deleting all the images
    $link = mysql_connect($host, $user, $pass)
      or die(mysql_error());
    mysql_select_db($db) or die('<response><sms>Could not select database</sms></response>');

          $query = "delete from images";
    $resultset = mysql_query($query);
    mysql_close($link);
```

```
    // Add the action to the actions table
    $link = mysql_connect($host, $user, $pass)
       or die(mysql_error());
    mysql_select_db($db) or die('<response><sms>Could not select database</sms></response>');

            $query = "insert into actions(action) values ('reset')";
    $resultset = mysql_query($query);
    mysql_close($link);
  }
```

The client will periodically request collage_actions.php which just renders the actions list as XML. Only the latest actions are acted on by the client. The following code shows how the actions are rendered as XML for the client:

```
    // Connecting, selecting database
    $link = mysql_connect($host, $user, $pass)
       or die(mysql_error());
    mysql_select_db($db) or die('<response><sms>Could not select database</sms></response>');

    $query = "select id, action, options from actions where id > " . $id;
         $resultset = mysql_query($query); // TODO: where id > ?id

    echo "<response>\n";

    while ($row = mysql_fetch_assoc($resultset)) {
       echo "<do>\n";
       echo "  <id>" . $row['id'] . "</id>";
       echo "  <action>" . $row['action'] . "</action>\n";
       echo "  <options>" . $row['options'] . "</options>\n";
       echo "</do>\n";
    }

    echo "</response>\n";
    // Closing connection
    mysql_close($link)
```

## Some Gotchas on the Server Code

- Some web servers don't allow URL gets which is used by getimagesize. Adding the php.ini file sometimes will override this behavior.
- Some web servers reverse-proxy php pages. Setting the expiration parameters fixes this but it's an easy step to miss.
- If the XML is not well formed, the XHR object will choke. Make sure that you wrap your responses in parent tags.

## Client: Connecting image data from the service to a Windows developer preview client

The Windows developer preview client was created starting from the blank application JavaScript project. Some very generic HTML was added that would later be used to render the content. The following shows the basic HTML that contains the collage, title, and status area, and app bar:

```
<body>
   <div id="title"><h1>TwilioCollage</h1></div>
   <div id="statusDiv">Nothing to see here ... yet!</div>
   <div role="region" class="item" id="images_area"></div>
   <section role="main" aria-label="Collage Area">
   <div>
      <canvas id="collageArea" width="1024" height="768"></canvas>
   </div>
   </section>
   <div id="theAppBar" data-win-control="WinJS.UI.AppBar" aria-label="Command Bar" data-win-
options="{position:'bottom', transient:true, autohide:0, lightDismiss:false}" >
      <div class="win-left">
            <button id="home" class="win-command">
               <span class="win-commandicon win-large">&#xE10F;</span><span class="win-
label">Home</span>
            </button>
            <hr />
            <button id="glow" class="win-command">
               <span class="win-commandicon win-large">&#x2383;</span><span class="win-
label">Glow</span>
            </button>
            <hr />
            <button id="reset" class="win-command">
               <span class="win-commandicon win-large">&#x260E;</span><span class="win-
label">Reset</span>
            </button>
      </div>
   </div>
</body>
```

The default.js file contains some basic functions that will initialize the canvas, retrieve its 2D context, and start the animation loop. The following code shows how this is done in the initialize() method:

```
   function initialize() {
      // Canvas initialization
      canvasArea = document.getElementById("collageArea");
      context = canvasArea.getContext("2d");

      // width / height initialization
      if (typeof window.innerWidth != 'undefined') {
         // Standards
```

```
        canvasWidth   = window.innerWidth;
        canvasHeight  = window.innerHeight * .8;
    } else if (typeof document.documentElement != 'undefined' && typeof
document.documentElement.clientWidth != 'undefined' && document.documentElement.clientWidth
!= 0) {
        // Quirks
        canvasWidth   = document.documentElement.clientWidth;
        canvasHeight  = (document.documentElement.clientHeight * .8) - 80;
    } else {
        // Legacy
        canvasWidth   = document.getElementsByTagName('body')[0].clientWidth;
        canvasHeight  = document.getElementsByTagName('body')[0].clientHeight * .8;
    }

    canvasArea.width  = canvasWidth;
    canvasArea.height = canvasHeight;

    // call yourself based on context (3rd party compat)
    if (window.requestAnimationFrame) {
        window.requestAnimationFrame(renderLoopRAF);
    }else if (window.msRequestAnimationFrame) {
        window.msRequestAnimationFrame(renderLoopRAF);
    }else if (window.mozRequestAnimationFrame) {
        window.mozRequestAnimationFrame(renderLoopRAF);
    }else if (window.webkitRequestAnimationFrame) {
        window.webkitRequestAnimationFrame(renderLoopRAF);
    }
  }
```

Once you have your canvas set up, you're ready to synchronize the images or perform other client actions.

### Retrieving and Rendering the Images

We currently have a functioning canvas but no images.  When the application starts up, it sets up an XML HTTP Request using the following call:

```
  function syncImages() {
    var URI = svcBaseURL + "collage_images.php";

    // TODO: startup code here
    if (maxId > 0) {
        URI += "?id=" + maxId.toString();
        statusDiv.innerText = "Adding new images from server ... ";
        WinJS.xhr({ url: URI }).then(processNewImages, downloadError);
        setTimeout(function (){ syncImages(); }, 5000);
    }else {
        statusDiv.innerText = "Batching images from server ... ";
```

```
        WinJS.xhr({ url: URI }).then(processImages, downloadError);
        setTimeout(function (){ syncImages(); }, 5000);
    }
  }
```

When **WinJS.xhr(….)** method is called, the function **processNewImages** is called on success, or **downloadError** on failure. Let's take a closer look at **processNewImages**. The **processNewImages** function will handle a request object, extract the twImage elements from the server, and will then add the images to arrays used for rendering. The following code shows how this is done:

```
  function processNewImages(request) {
    var items = request.responseXML.selectNodes("//twImage");
    var status = "Images synchronized and rendered.";

    var id = -1;

    if (items.length < 1) {
      statusDiv.innerHTML = "No new images currently, text: " + twilioNum + " with the message "+
twilioExt + " <i>ImageURL</i>";
    } else {
      for (var i = 0; i < items.length; i++) {
        var item = items[i];

        id = parseInt(item.selectNodes("id")[0].text, 10);
        var src = item.selectNodes("src")[0].text;
        var alt = item.selectNodes("uploadedBy")[0].text;

        // assumption: no duplicate IDs, ascending order
        if (id > maxId) {
          maxId = id;
          addImage(id, src, alt);
        }

      }
      statusDiv.innerText = status;
    }
  }
```

Add image appends the images to the hidden DIV that holds them and then adds image details to an array with position information for animation and rendering. If you look at the DrawCollage function, you can see how the images are rendered.

### Resetting the Images

To ensure that the client resets images when the collage is reset, the Windows developer preview client again uses the XHR request to traverse a list of "Actions" from the web. The server creates an XHR

request to the collage_actions.php page which then sends it XML with “actions”. If the action is reset, the client removes all of the images in its hidden div and clears the list of images used in the render loop. For brevity, the XHR code isn’t shown in full, but the following section shows where the reset function is called from within a traversal of the action nodes:

```
function processNewActions(request) {
   var items = request.responseXML.selectNodes("//do");

   var id     = -1;
   var action = "";

   if (items.length == 0) {
   } else {
      for (var i = 0; i < items.length; i++) {
         var item = items[i];

         id     = parseInt(item.selectNodes("id")[0].text, 10);
         action = item.selectNodes("action")[0].text;

         if (action == "reset") {
            resetImages();
         }

         // assumption: no duplicate IDs, ascending order
         if (id > maxAction) {
            maxAction = id;
         }
      }
   }
}
```

### Triggering the Reset IVR from the Windows developer preview

To trigger the same reset service as is initialized through sending a text message to the service with ‘reset’, yet again the XHR object is used. The REST URI is formed for a “call” command and then parameters are set up for the call. After that, the XHR is initiated and if the function succeeds, Twilio will trigger the collage reset. The following code shows how this is done.

```
function callAndReset() {
   // URI string building
   var URI      = "";
   var baseURL  = "https://api.twilio.com/2010-04-01/";
   var acctPath = "Accounts/" + acctSID + "/";
   var action   = "Calls";

   // Call handler URL
   var callHandler = svcBaseURL + "handle_call.php";
```

```
    var params   = "From=" + fromNum;
    params     += "&To="   + toNum;
    params     += "&Url="  + callHandler;

    URI = baseURL + acctPath + action;

    WinJS.xhr({
      type: "POST",
      url:  URI,
      user: acctSID,
      password: token,
      headers: { 'Content-type': 'application/x-www-form-urlencoded'},
      data : params
    }).then(
      function (request) {
        // Nothing to do here...
      },
      function (request) {
        // HANDLE error case here or, could just retry...
        // setTimeout(function (){ callAndReset(); }, 10000);
      }
    );
}
```