**Requirements**

Create a DisplayImageComparerActivity in Granite that compares the phone display image with a reference image (local file).

Browse button

Display file opening dialog

Display selected image name in a textbox.

Display selected image in masking panel.

Capture button

                Take capture from the phone and display it in masking panel.

Save button

1. Opened image:

Merge masks in the image.

Save with existing filename.

1. Captured image:

Merge masks in the image.

Display save file dialog.

Display filename in a textbox.

Masking panel

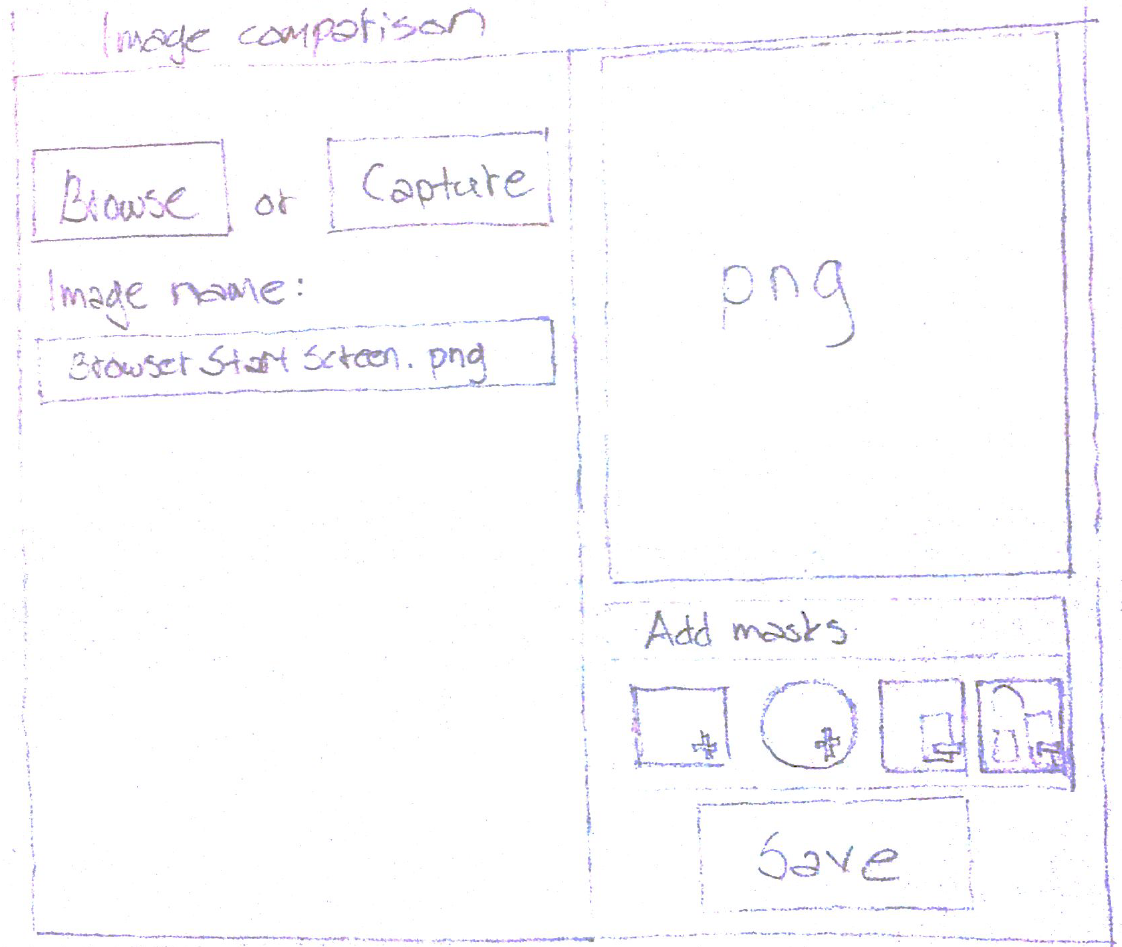
                Rectangle or circular masks can be applied to image.

                Masks on image should be able to be overlapped and resized.

                Remove selected masks or remove all masks.

Note:

Scanned UI drawing:



Then we would need the framework support for the actual comparison.

Reporting should also be planned.  
Requirement from Timo is to have file-based reference image comparison that does not use the cloud database in any way.

Comments from Mark and Jani (18/07/2012):  
With it being file-based in Granite, I think it would be easier for users if we only have a single image, otherwise I could see problems where users copy the reference image file but forget the masked XML meta-data file, but I am now concerned whether or not we can easily achieve this.

Yeah, good point. We should try to keep it as a single file if possible.

1. Insert metadata about the masks within the image data. Don’t know if this is possible and how much data can be saved. Also this may get quite tricky working with the byte data.
2. Provide Unmask operation instead of remove mask. Existing united mask shape can be modified by unmasking certain areas with the same rectangle and circular shapes.

Just one other thing, do you want the spinning image eye candy in Granite or not?   
I assume not, since you only show a single PNG.

Yep, at first I thought it’s not needed and it’s better to show the masks straight away.

But on second thought in some cases the user may want to view the original image also.

Let’s do the flip if it isn’t too hard to implement.

03/08/2012: It is more work to implement this in WPF compared with Silverlight, so we have decided not to provide this in the initial version.

**Actions**

Action: Wekey to do Activity and UI (ActivityDesigner). Juhani to review.

~~Action: Find out if there is a WPF version of the Silverlight control used in STE.  
 Assigned to Wekey.~~

DONE: 03/08/2012

~~Action: Mark to transfer Granite UI source code to Wekey.~~

DONE: 20/07/2012

~~Action: Investigate whether XML meta-data could be stored inside an image (e.g. Image.Tag) and whether there are any limitations (e.g. limit on amount of data stored).  
Decides whether we implement Option B or C.~~

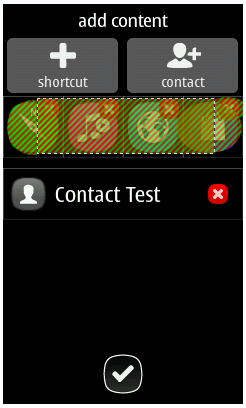
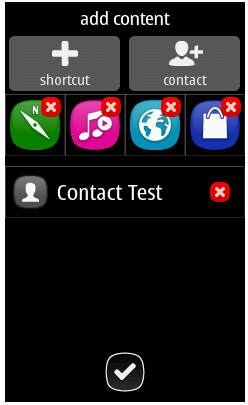
~~Assigned to Wekey.~~

DONE: 20/07/2012

CONCLUSION: Image.Tag does not work but can store in unlimited text in property item of image.  
DECISION: Add the mask information as a property item of the image.

~~Action: Better quality scanned image or the UI drawing (Jani). Provide Wekey with a good picture to show the UI.~~

DONE: 20/07/2012

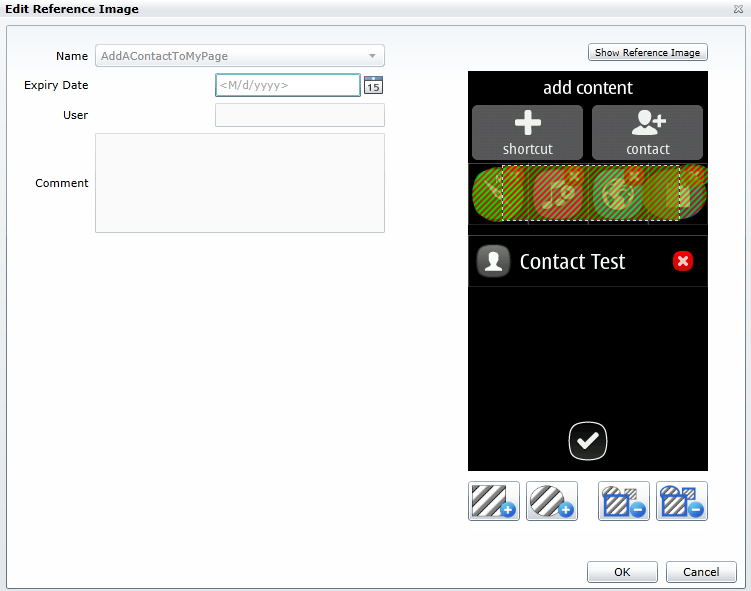
**Option A – Separate files (Rejected)**XML Mask Meta-Data

GUI writes

Activity / framework reads

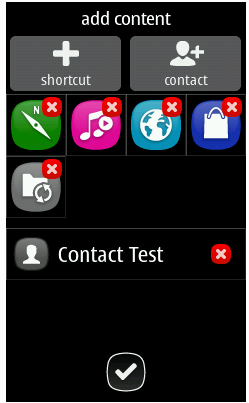
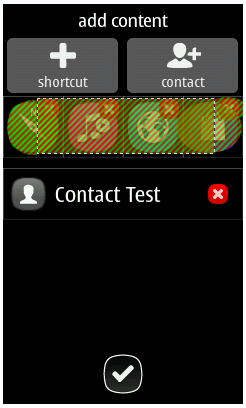
GUI reads

GUI reads



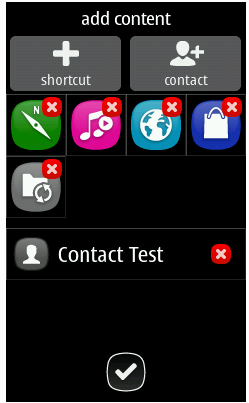
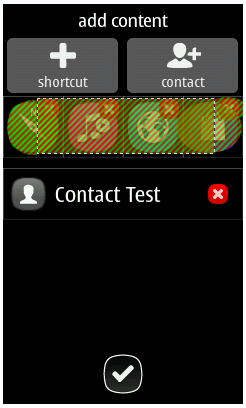
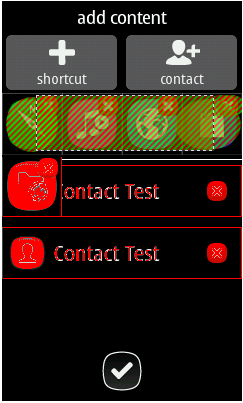
1. User adds reference image and mask

2. Activity runs in Granite Framework

Actual phone image Reference image

**3. Framework generates images for report**

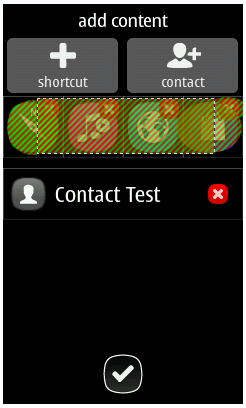
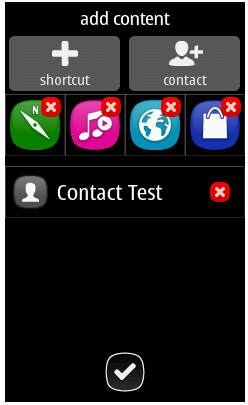
Actual phone image Reference image Differences

Disadvantages:

Too many files, i.e. original image, masked reference image (with Alpha changed and masked areas coloured in), XML mask meta-data

Advantages:

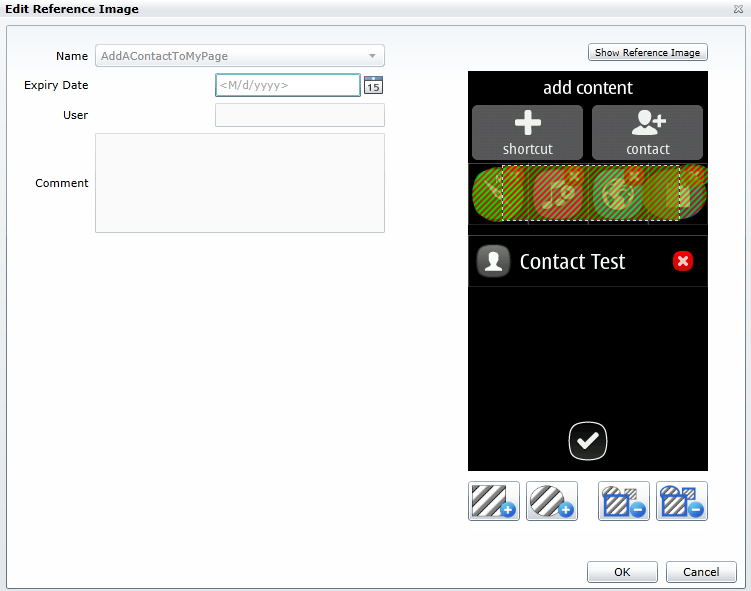
Very easy for the Framework since the only image that needs to be produced is the Differences image.

**Option B – Not storing XML Mask Meta-Data (Rejected)**

GUI writes

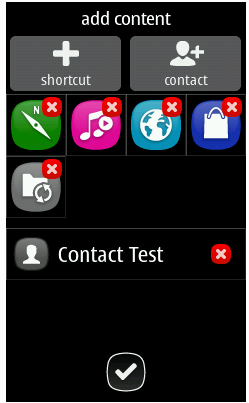
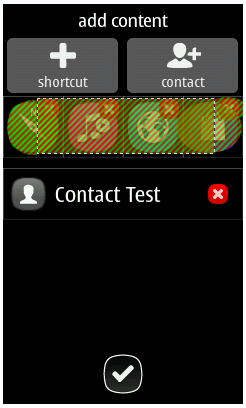
Activity / framework reads

GUI reads



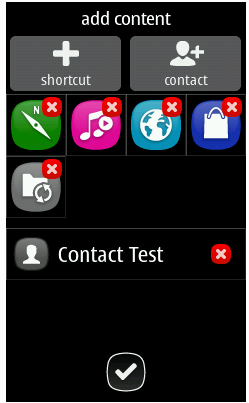
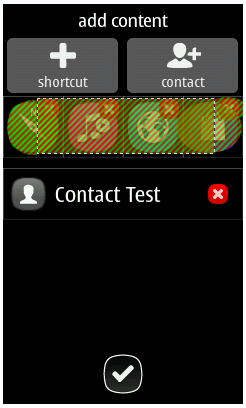
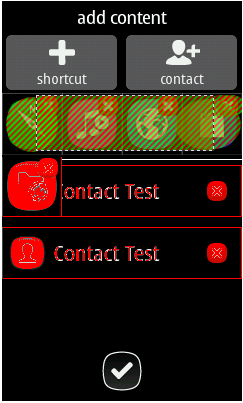
1. User adds reference image and mask

2. Activity runs in Granite Framework

Actual phone image Reference image

**3. Framework generates images for report**

Actual phone image Reference image Differences

Disadvantages:

There is no way to get back original XML mask meta-data, so need a mechanism to erase areas of the mask, since we are only working on Alpha values.

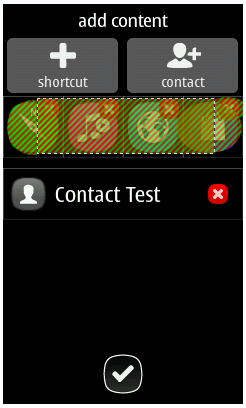
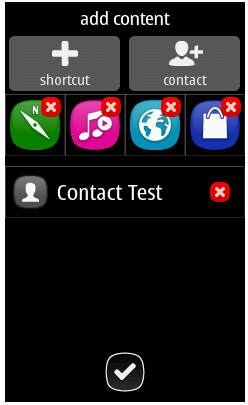
UI will need to colour in mask area by checking Alpha values.

Framework will need to colour in mask area for the report in order to produce Reference image, as well as the Differences image.

Advantages:

Only require a single reference image file (with different Alpha values for mask area).

Note: UI can still show original image by resetting all Alpha values, as well as produce image with mask.

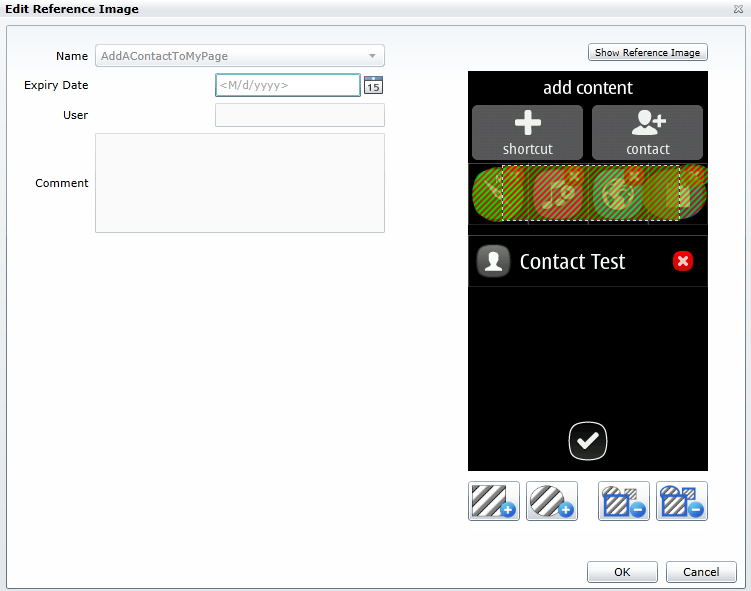
**Option C – XML Mask Meta-Data stored inside image (Accepted)**XML Mask Meta-Data

GUI writes

Activity / framework reads

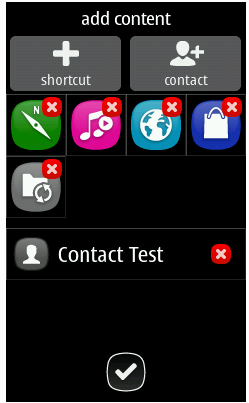
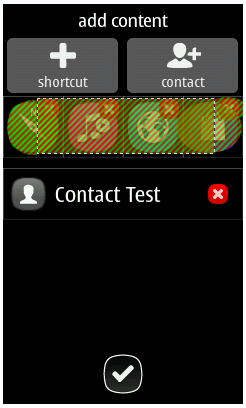
GUI reads

GUI reads



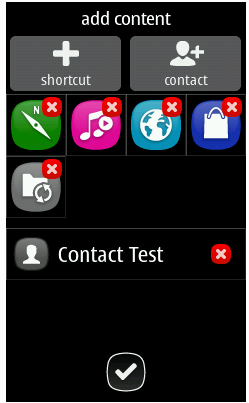
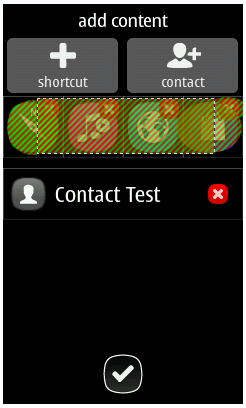
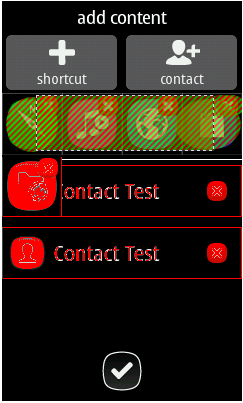
1. User adds reference image and mask

2. Activity runs in Granite Framework

Actual phone image Reference image

**3. Framework generates images for report**

Actual phone image Reference image Differences

Disadvantages:

UI will need to colour in mask area by using XML meta-data.

Framework will need to colour in mask area (using XML meta-data) for the report in order to produce Reference image, as well as the Differences image.  
  
Advantages:

Only require a single reference image file (with XML mask meta-data as an image property).

Produces good UI where the user can remove any individual masks that were previously added. Do not need to have an erase capability, since the individual masks can be selected and removed.

**Issues:**

1. Reference image file locations

You can currently load and run any Studio workflow XAML file located anywhere on your PC.  
If an activity is now linked with a reference image file path, that won't really work. Also, if a user is busy changing a Studio test and has not saved it, when the user runs this test, it is actually saved to a temporary directory. For this reason, relative reference image files paths would not work.  
  
It could work using absolute file paths, but then problems occur if different users run Granite or studio tests from a different directory structure.  
  
One solution could be for the activity to store the image as its own property so that it is part of the XAML. The image  
  
Note there is also support in STE to use local reference images, but it is very limited (i.e. absolute path for file must not change and no support for masking).

Solution: Always only store the reference image file name in the activity

~~“..\reference\_images”~~

~~Use this fixed place under Granite directory. Both UI and Framework can refer to that path with the same notation.~~

“..\reference\_files\images”

Use this fixed place under Granite directory. Both UI and Framework can refer to that path with the same notation.

Limitations of using local reference image files  
  
Requirement is to only port the DisplayImageComparerActivity STE code that compares locally stored images, NOT any of the cloud and database functionality.  
  
Having looked at the solution in Granite where we use locally stored reference image files, I am concerned that the display image comparison implementation in Granite will not be as sophisticated as that provided by STE. For instance there will either be no support or limited support for the following:

1. Multiple reference images where a phone display image changes over time and we can easily store reference images for all the variations. This is difficult and potentially problematic with a file based solution. This may require us to create directories to group all reference image files with all variations, or come up with some file naming convention for multiple reference images.
2. The ability to have the workflow XAML tests work on any products, since with local reference image files it will be extremely difficult to maintain and partition all phone reference images per product, or ensure that the XAML test files are distributed with the correct set of phone reference images.
3. Ability to easily e-mail or copy and run workflow XAML tests amongst all users across multiple sites (since users will also need to remember to copy across all associated phone display reference images and they must be stored in a specific directory on the local PC, rather than say a shared network folder or central cloud database).
4. Easily store and maintain additional information with reference images, such as expiry dates for certain reference images. We would either end up with lots of files for users to maintain or we need to be able to ensure that we can always store all required additional information inside the reference image file, as we have done for the image mask information.
5. Allow users to very easily maintain and update all of the reference images. The workflow activity will be linked to a single reference image file therefore the reference image will need to be maintained from within the test case.
6. Mechanism where users can easily click a button when reviewing display image reference comparison results to update/upload a new reference image (such as is provided from the System Test Image Viewer). Currently there is no such mechanism planned for the report in Granite.
7. File based reference images does not scale very well compared to a database, if we have lots of different products with hundreds of reference images.

We can however provide a very simple implementation, where phone display comparison is done with a single masked reference image file.

1. Framework image comparison implementation  
     
   There are a number of options:
2. Take the image comparison from STE and implement a simplified version in Granite DLL, so Framework could call the DLL to use this functionality (no workflow required)
3. C# code could be converted to IronPython code
4. Framework developers could implement their own solution entirely (or possibly use functionality they had in ART2)

/// <summary>

/// Compares two bitmaps and throws an <see cref="ImageDefectException"/> if they do not match.

/// </summary>

/// <param name="actualImage">Actual display image bitmap from the phone.</param>

/// <param name="expectedImage">Expected display image bitmap.</param>

/// <param name="imageName">Name for the image.</param>

/// <exception cref="ArgumentNullException">Thrown if any of the supplied image arguments are <c>null</c>.</exception>

/// <exception cref="ImageDefectException">Thrown if either the sizes of the two images differ or if the images themselves have any differences.</exception>

public static void Compare(this Bitmap actualImage, Bitmap expectedImage, string imageName);

1. Framework could use the cloud very easily at some point in the future:
   1. To get the product reference images would require the Framework to call the STE DLL that already has this functionality (no workflow required)  
      public System.Drawing.Image[] GetImages(string product, string branch, string imageName,   
      DecryptDelegate decryptDelegate = null);

/// <summary>

/// Retrieves the images for the specified product and branch and allows the image to be decrypted by supplying a delegate that performs the decryption.

/// </summary>

/// <param name="product">Product name (e.g. 'RM-72').</param>

/// <param name="branch">Phone branch (e.g. 'ICPR92').</param>

/// <param name="imageName">Name of the image (e.g. 'ClipArt1Added').</param>

/// <param name="decryptDelegate">Optional delegate that is used to perform the decryption.</param>

/// <returns>Array of images for the specified product and branch.</returns>

/// <exception cref="ArgumentException">Thrown if the <paramref name="product"/>, <paramref name="branch"/> or <paramref name="imageName"/> is not recognized.</exception>

/// <exception cref="DataException">Thrown if there is any logical fault in the data stored for the specified product and image.</exception>

public System.Drawing.Image[] GetImages(string product, string branch, string imageName, DecryptDelegate decryptDelegate = null);

1. STE DisplayImageComparerActivity allows users to optionally also specify the following:  
   Tolerance (Percentage 0% to 100% where we can allow for some “acceptable” variation).  
   Red, Green, Blue ranges to allow some minor variation in red, green and blue values.  
     
   No plans in first phase to have this level of functionality in Granite.
2. Storing reference images inside the workflow XAML

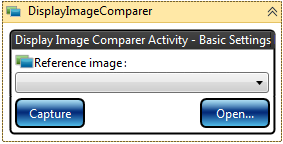
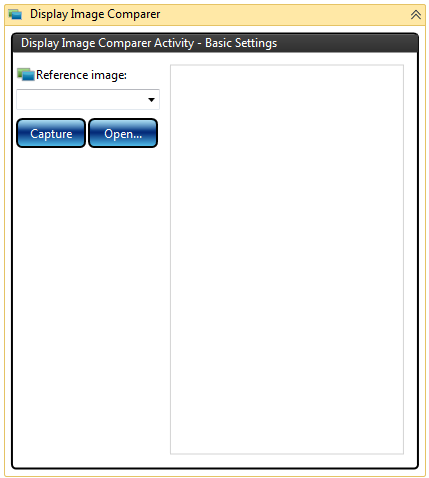
One possible solution could have been to store the reference image itself (i.e. Bitmap) as a property on the DisplayImageComparerActivity. This has the advantage that if a user copies the XAML test case to send to another user, it contains any reference images.

The problem though is that you cannot easily re-use that reference image from other test cases. The other problem is that if there are lots of test cases that need to use the same reference image and that reference image needs to be changed then you would have to update all the tests, rather than simply replace the one reference image. From experience of image comparison in STE, users typically re-use reference images across multiple tests. This idea has therefore been rejected.

**User interface and functionality**

All functionality to create and maintain the reference images with masks should be made available from within the ActivityDesigner. The ActivityDesigner should not provide the functionality by displaying additional dialogs.

The user drags the DisplayImageComparerActivity from the toolbox onto the workflow designer. At this stage no reference image has been selected, loaded or captured, so it should present a UI without an image.

 **Or**

From here the user can either:

1. Select an existing reference image (which may have masks) using the names in the dropdown list.  
   Note: users should not be allowed to type in
2. Open any image file on their local PC.  
     
   Note: If a reference image name had previously been selected in the dropdown list, then after the image has been loaded via the Open functionality, then the selected item in the reference image names dropdown should be cleared (i.e. no reference image name selected), since we have loaded a new image and we do not yet know the reference image name that it will be given.
3. Capture an image from the current phone.  
     
   Note: If a reference image name had previously been selected in the dropdown list, then after the image has been loaded via the Capture functionality, then the selected item in the reference image names dropdown should be cleared (i.e. no reference image name selected), since we have loaded a new image and we do not yet know the reference image name that it will be given.

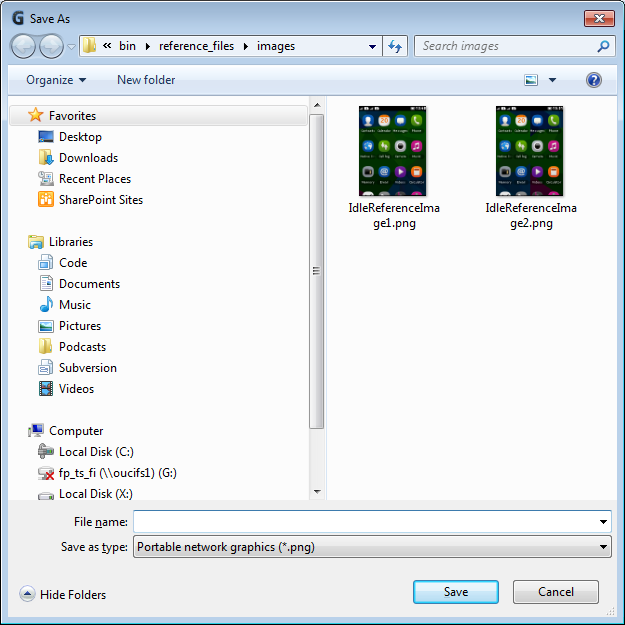
Once we have an image from any of the above sources, we can then display the image in the MaskEditorControl:



From here a user can add or update any masks on the image. To save their changes, the user presses the Save button.

If we already have a reference image name (i.e. that is scenario 1) in the list above), then we simply overwrite the reference image file (in the reference images directory) using the name that was selected in the dropdown list.

If we do not have a reference image name (i.e. that is scenario 2) or 3) in the list above), then I think the simplest option is to pop up a SaveFileDialog with the InitialDirectory property set to the directory where we store the reference image file:



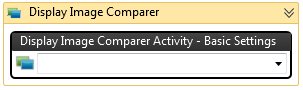
If the user chooses to put the file in a different directory, then the code should display an error message and not save the reference image file.

Once the reference image file has been saved, the name should appear here:



If the user selects New or Open from Studio to open another test and then opens this original test again, then the DisplayImageComparerActivity should appear as above with the reference image name displayed together with the reference image (with any masks applied).

All of the above screenshots show when the activity is Expanded. If the activity is Collapsed it should appear similar to this:



**IronPython Scripting Solution**

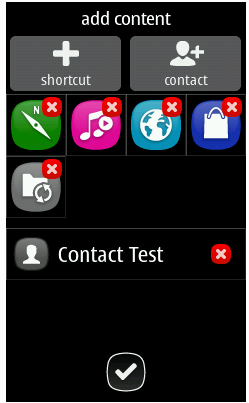
There may be a requirement at a later date to be able to perform reference image comparison from within IronPython scripts. As a result to make this easier, there may be a need to create a tool that helps the developers that write test scripts, to be able to create and maintain reference images using masks. If this requirement arises, then we may develop the functionality so it can be made available through a separate dialog, for instance in a similar way to how the Capture Tool helps the test script developers create IronPython code that uses the display elements. This reference image comparison tool would most likely make use of the same controls that were used in the ActivityDesigner.

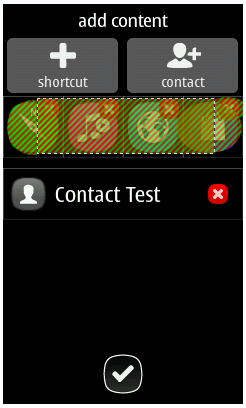
Currently though, the creation and maintenance of the reference images with masks shall only be made available from within Studio. The functionality to perform a comparison with a reference image mask will be available from both Studio and IronPython scripts, since the image comparison functionality will be implemented in the Framework with an IronPython call, similar to the following:

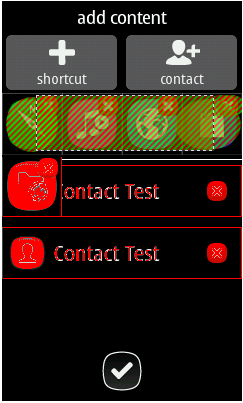
self.compare(referenceImageName)

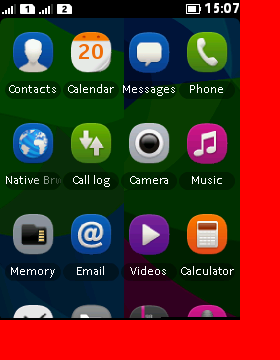
For the report the Framework should generate three images:

1. The actual phone display image.



1. The reference image with the mask areas shown.  
   Note that the original reference image will only have the mask areas as XML meta-data, so it is necessary for these areas to be coloured in, in order to show them to the user:  
   
2. The differences image with the differences marked in red (as well as any masked areas shown):



The comparison algorithm should cater for situations where the actual phone display image size is different to the reference image size. This should either generate a meaningful error message and/or generate a differences image showing the user that the size is different (marked in red):  


**Overview**

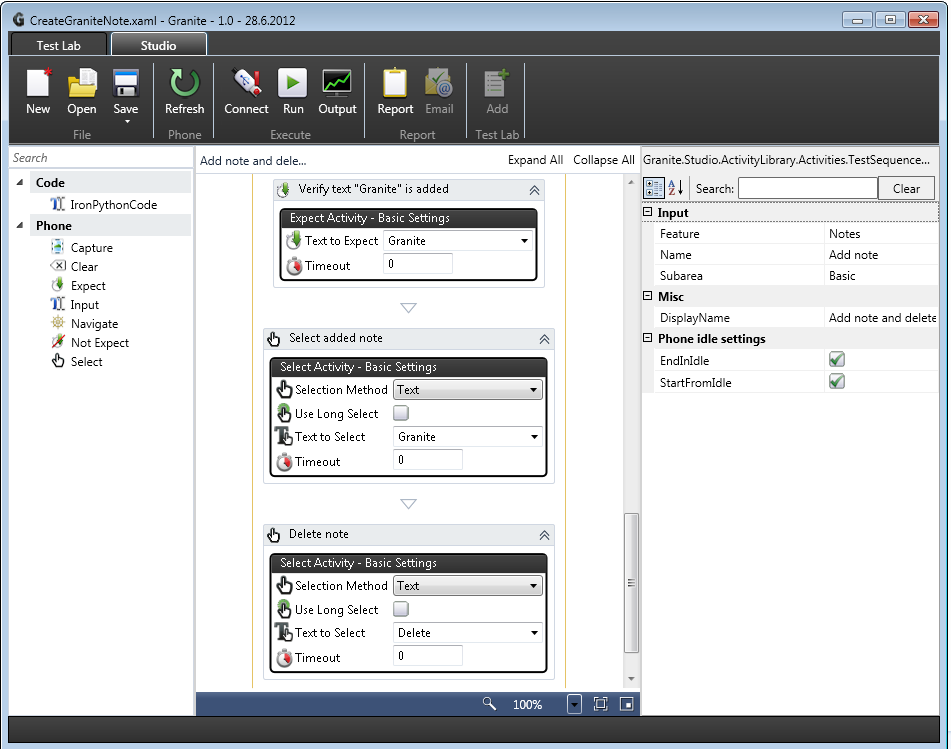


Figure 1 Existing Granite Studio UI and workflow activities

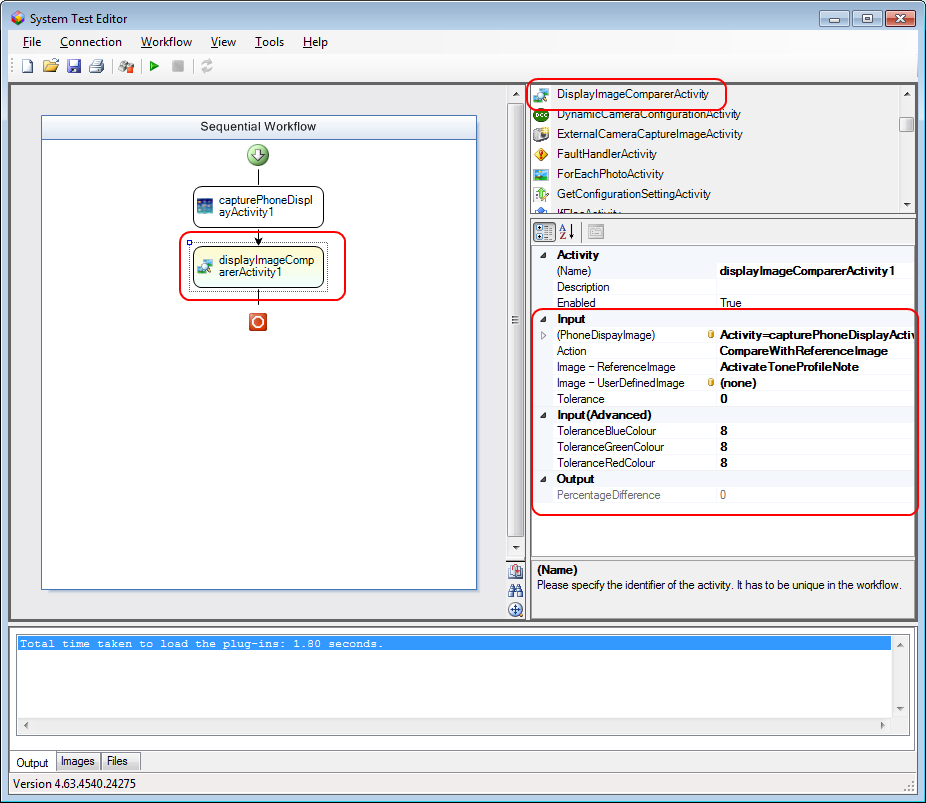


Figure 2 STE with DisplayImageComparerActivity

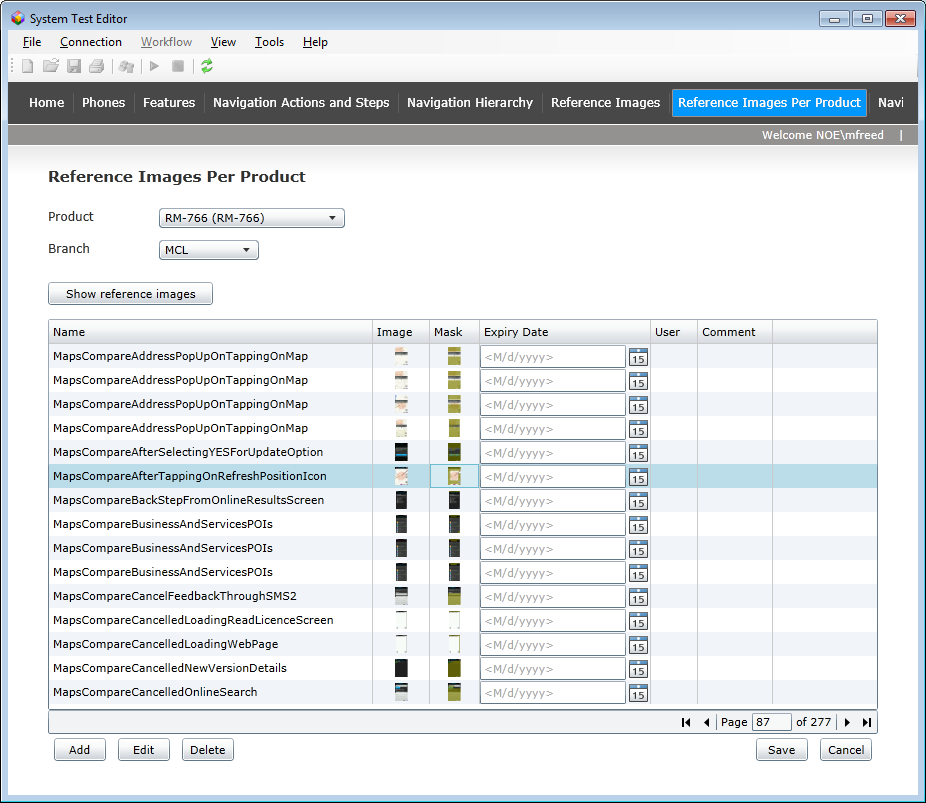
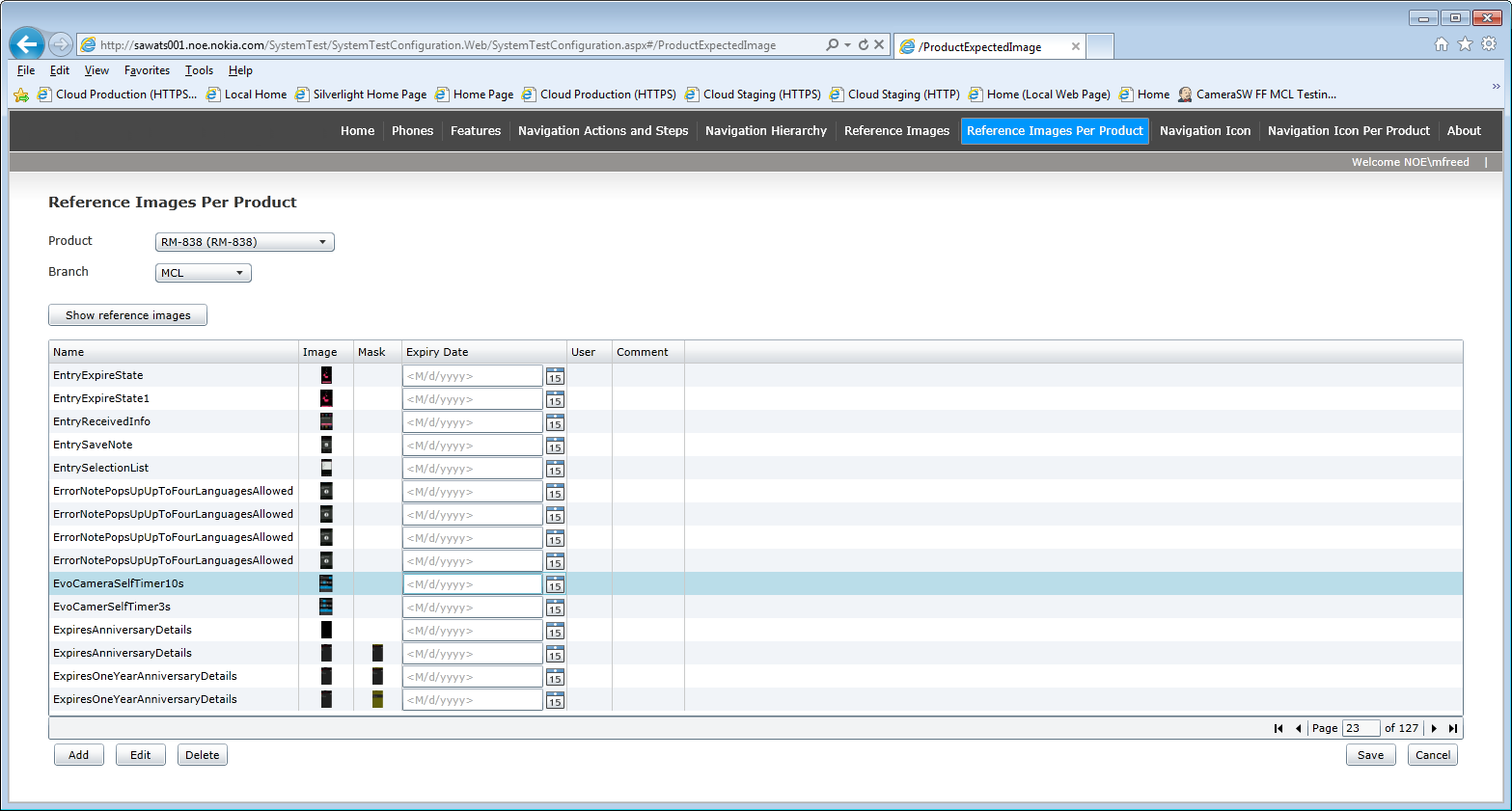
Currently in STE users maintain the reference images and masks through web pages (hosting browser control in STE):

Figure 3 STE with web pages to maintain reference images

Reference images together with any masks and an expiry date are held in a cloud database. They are partitioned per product, so users can view or maintain reference images on a per product basis.

STE users can also access the web pages to maintain reference images directly from Internet Explorer:

From Granite we need functionality to allow users to add local reference images and masks, similar to the following functionality provided by the STE web pages: