**Requirements**

The requirement is to update the Display Image Comparer activity that compares the phone display image with a single reference image (local file), so that it can compare the phone display image with multiple reference images and allow the user to manage multiple reference images.

**Existing DisplayImageComparerActivityDesigner**

The current DisplayImageComparerActivityDesigner only allows a single image to be stored for a specified reference image name:



Figure 1 Current Display Image Comparer Activity (in Granite 1.0.5)

The reference image name correlates to a single reference image file name. For instance, in the example screenshot above the reference image name “Image002IdleWithMask” will show the “Image002IdleWithMask.png” reference image file.

**DisplayImageComparerActivityDesigner Changes**

In order to be able to iterate through reference images as well as add or delete individual images, we ideally need to be able to implement or use something very similar to the DataForm control that is available in Silverlight (see DataForm sample from the Silverlight Toolkit Samples: <http://www.silverlight.net/content/samples/sl4/toolkitcontrolsamples/run/default.html>). This control allows users to iterate through the data entries, as well as add new entries (using the plus sign) or delete particular entries (using the minus sign). Click on the above link to interactively use this DataForm Silverlight control.

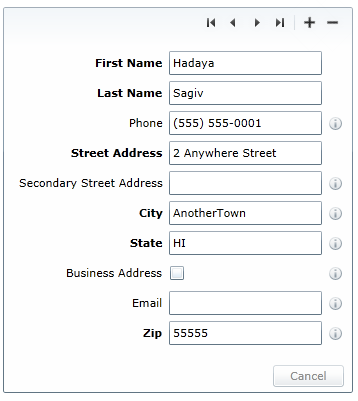


Figure 2 Silverlight DataForm

This would allow us to provide a user interface where users could iterate through each of the reference images as well as add or delete any of the reference images.



Figure 3 DisplayImageComparerActivityDesigner with control similar to Silverlight DataForm

When the Display Image Comparer activity executes, we need to be able to refer to each of the reference images by name, but since they all are listed under a single reference image name, we need to provide the user with their index number, so that we can distinguish between them and refer to them correctly in the test report. For this reason we should display the index number of the reference image to the user as shown below:



Figure 4 DisplayImageComparerActivityDesigner showing index

Users could then iterate through each reference image, e.g. ‘1 of 3’, ‘2 of 3’ and ‘3 of 3’. We can also refer to the particular reference image in the test reports, e.g. ‘2 of 3’.

If the user was say on image ‘2 of 3’ and pressed the minus button to delete this image, the ‘2 of 3’ image file would be deleted and the ‘3 of 3’ image would be shown, however the index and total number of images would have changed, therefore ‘2 of 2’ would then be displayed since there are only two reference images and the user is viewing the last of the two.

Adding multiple reference images requires some changes to the existing user interface. Clicking the Capture or Open button in the existing DisplayImageComparerActivityDesigner did the following:

It cleared the Reference Image name and displayed the new reference image in the MaskEditorControl, where the user could press Save and enter in the filename for the reference image.

This functionality will not work now that we are implementing multiple reference images, since we need separate functionality to:

1. Create a new reference image name.
2. Add an image (using Capture or Open to obtain the new image), update an image (using Capture or Open to obtain the new image) or delete an image.

Since we need to have this functionality separated, we will need to modify the button layout as follows:



Figure 5 New DisplayImageComparerActivityDesigner with updated button layout

**Use Case**

Assuming we have no reference images on the PC and the user drags the Display Image Comparer activity onto the workflow designer, they should be presented with the following:

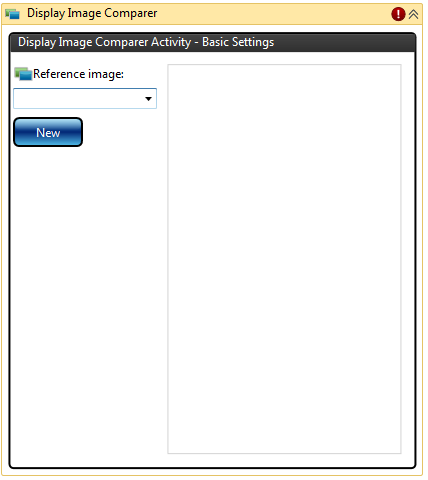


Figure 6 New DisplayImageComparerActivityDesigner with no reference image name selected

No reference image names will be available, since none have yet been created. To create a new reference image name the user presses the New button.

They should then be presented with a modal dialog box prompting them for the reference image name.

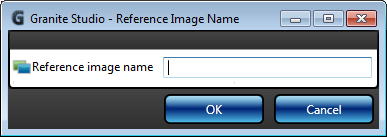


Figure 7 Dialog prompting user to enter in the a reference image name

***Topic for discussion****: Another option could be for the user to enter in the name and then press the New button, but this may not be as intuitive for users:*

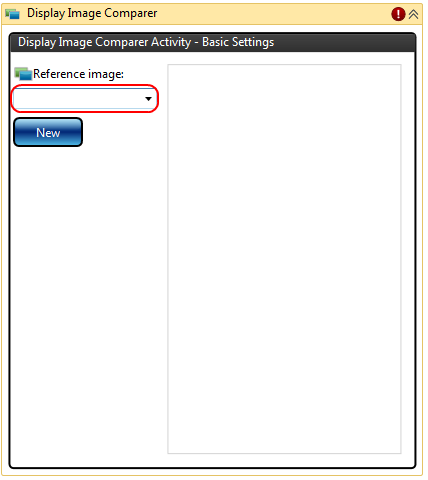


Figure 8 Alternative option where user enters in the reference image name before pressing New button

*Alternatively we could allow the user to Capture or Open the image and then get them to fill in the filename in the SaveFileDialog, and use the filename as the reference image name, however since we are now keeping multiple reference images this could lead to problems if users type in a filename ending with ~1, ~2, ~3, etc.*IdleScreen.png  
Calculator.png  
Calculator~2.png  
Calculator~3.png  
BrowserGoogle.png

*The recommendation is therefore to introduce the modal dialog to prompt the user for the new reference image name, but this needs to be discussed amongst the Granite GUI team.*

After the reference image name has been specified, the user presses the Capture or Open button:

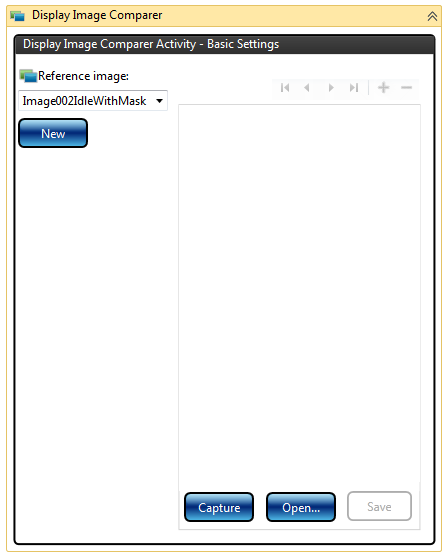


Figure 9 Adding new reference image

The Save button is disabled (as shown in Figure 9) until an image has been captured or opened.

Note that since we are in the process of adding a new image during this stage, this control will be disabled:  


The user can then apply any required masks:



Figure 10 Adding reference image masks

The user would then press the Save button:



Figure 11 Reference image saved

Note that there is no need to present a SaveFileDialog since we have the reference image name and the directory for the reference images is a fixed location.

Once the reference image has been saved the new index should be displayed (e.g. ‘1 of 1’) and this control should be enabled:  
 

Note that the various arrows that allow you to iterate through the images, will be enabled/disabled depend on whether the user is viewing the first, middle or last image.

From here the user can add another reference image (with the same name) by pressing the plus sign and they would then be presented with Figure 9.

Note that if the user presses the minus button repeatedly until they are on image ‘1 of 1’ and then they press the minus button again, they should be presented with Figure 9 so that they can add a single image.

**Image Comparison Algorithm Changes**

The algorithm for performing the image comparison needs to be modified.

The existing logic was as follows:

1. Compare the actual display image with the reference image (that may contain masks).
2. Produce a difference image (with the masks drawn) and return a flag that indicates whether the actual display image matched the reference image or not.
3. If the image matched, the test report XML information contains the actual image and the reference image (including the reference image name).
4. If the image did not match, the test report XML information contains the actual image, the difference image and the reference image (including the reference image name).

For multiple reference images, the new logic is as follows:

1. If there is only one reference image, use the same existing logic as described above.
2. If there is more than one reference image, calculate the percentage difference (excluding masked areas) between the actual image and each reference image.
3. For the first comparison where there is a 0% difference (if there is one) the test report XML information should contained the actual image and the reference image (including the reference image name **together with its index, e.g. ‘2 of 3’**).
4. If none of the reference images match, the test report XML information should contain the actual image, the difference image **produced from the reference image that had the closest match** and **all** references image (including the reference image name **together with its index, e.g. ‘2 of 3’**).

***Topic for discussion****: STE allows a user to enter in a Tolerance percentage, where users can specify an acceptable variation as a percentage. A Tolerance value of 0% (which is the default) means an exact match is required (excluding masked areas). A Tolerance value of 100% would cause the actual image to match any reference image. ATolerance of say 10% means it would match if up to 10% of the pixels (excluding masked areas) differ. Since as part of the multiple reference images feature we are implementing in Granite, we are already calculating the percentage difference in order to find the closest match, we could later on relatively easily add the Tolerance property to the Display Image Comparer activity. Adding the Tolerance property may however require more work to be done for the XML report information and displaying the report, since this tolerance information would need to be made clear to the user.*

**Reference Image File Name Changes**

The current reference images are stored under the \reference\_files\images\ directory. This does not change for the multiple reference image functionality; however the logic for retrieving the list of reference image names needs to be changed slightly, as well as adding logic for storing the multiple reference image files.

Let’s assume we had the following reference image file under the \reference\images\ directory:

IdleScreen.png  
Calculator.png  
BrowserGoogle.png

The reference image names displayed by the activity would be “IdleScreen”, “Calculator” and “BrowserGoogle”.

If two multiple reference images were then added for Calculator, the following files would then exist:

IdleScreen.png  
Calculator.png  
Calculator~2.png  
Calculator~3.png  
BrowserGoogle.png

The reference image names displayed by the activity would still be “IdleScreen”, “Calculator” and “BrowserGoogle”.

As shown above, the logic for retrieving the list of reference image names needs to exclude any files with the name \*~***d***.png where ***d*** can be any number of digits.

If the users was viewing the Calculator~2.png image from the activity designer (i.e. image ‘2 of 3’) and pressed the minus button to delete it, the Calculator~2.png will be deleted and the Calculator~3.png image would be renamed to Calculator~2.png. The activity designer would then show the new Calculator~2.png image (i.e. image ‘2 of 2’).

**Report XML changes**

When the existing Display Image Comparer activity in Granite compares the phone display image with a reference image it produces XML using a single reference image. The XML produced is detailed below:

In the situation where the current display image matches the reference image, the following of XML format is produced containing information about the 2 images:

<teststep name="Display image comparison teststep">

<displayimagecomparison name="Display image comparison" result="Passed" >

<image name="Actual image" type="ActualImage" filename="capture\_3.png" />

<image name="'Idle' reference image" type="ReferenceImage" filename="capture\_3\_reference.png" />

</displayimagecomparison >

</teststep>

In the situation where the current display image does not match the reference image, the following XML format is produced containing information about the 3 images:

<teststep name="Display image comparison teststep">

<displayimagecomparison name="Display image comparison" result="Failed" >

<image name="Actual image" type="ActualImage" filename="capture\_3.png" />

<image name="Image showing differences with the reference image" type="DifferenceImage" filename="capture\_3\_difference.png" />

<image name="'Idle' reference image" type="ReferenceImage" filename="capture\_3\_reference.png" />

</displayimagecomparison >

</teststep>

For multiple reference images, in the situation where the actual display image matches any one of the multiple reference images, the following XML would be produced, containing information about the actual display image and the particular reference image it matched, including its index number in the reference image name so that it can be uniquely identified:

<teststep name="Display image comparison teststep">

<displayimagecomparison name="Display image comparison" result="Passed" >

<image name="Actual image" type="ActualImage" filename="capture\_3.png" />

<image name="'Idle **(2 of 3)**' reference image" type="ReferenceImage" filename="capture\_3\_reference.png" />

</displayimagecomparison >

</teststep>

For multiple reference images, in the situation where the actual display image does not match any of the multiple reference images, the following XML would be produced, containing information about the actual display image and the references images that did not match, together with a difference image showing the differences with the reference image that had the closest match:

<teststep name="Display image comparison teststep">

<displayimagecomparison name="Display image comparison" result="Failed" >

<image name="Actual image" type="ActualImage" filename="capture\_3.png" />

<image name="Image showing differences with the 'Idle **(2 of 3)**' reference image that had the closest match" type="DifferenceImage" filename="capture\_3\_difference.png" />

<image name="'Idle **(1 of 3)**' reference image" type="ReferenceImage" filename="capture\_3\_reference.png" />

<image name="'Idle **(2 of 3)**' reference image" type="ReferenceImage" filename="capture\_3\_reference**~2**.png" />

<image name="'Idle **(3 of 3)**' reference image" type="ReferenceImage" filename="capture\_3\_reference**~3**.png" />

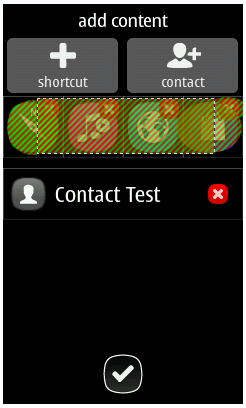
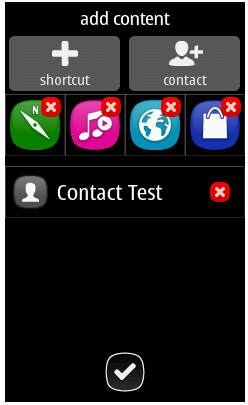
</displayimagecomparison >

</teststep>

***Topic for discussion****: The above XML does not provide any mechanism in the report to be able to correlate which reference image the difference image was produced from, although the name does inform the user which reference image it was generated from. For report display purposes, if we need to be able to correlate this, we may need to add an* id *number attribute to the XML for the images, and add a* reference\_id *attribute to the difference image so it is easy to correlate the images. If we decide to do this, then we should also consider whether we still use the* <image> *XML node name for all of these types of images, or whether we should create separate XML nodes which have different attributes, e.g.* <actualimage>*,* <differenceimage>*,* <referenceimage>. *So for instance a* <referenceimage> *would have an* id *attribute and a* <differenceimage> *would have* reference\_id *attribute.*

**Appendix A – Background information on original Display Image Comparer Design**

This section is provided for background information on the existing Display Image Comparer functionality as well as some background information on STE.

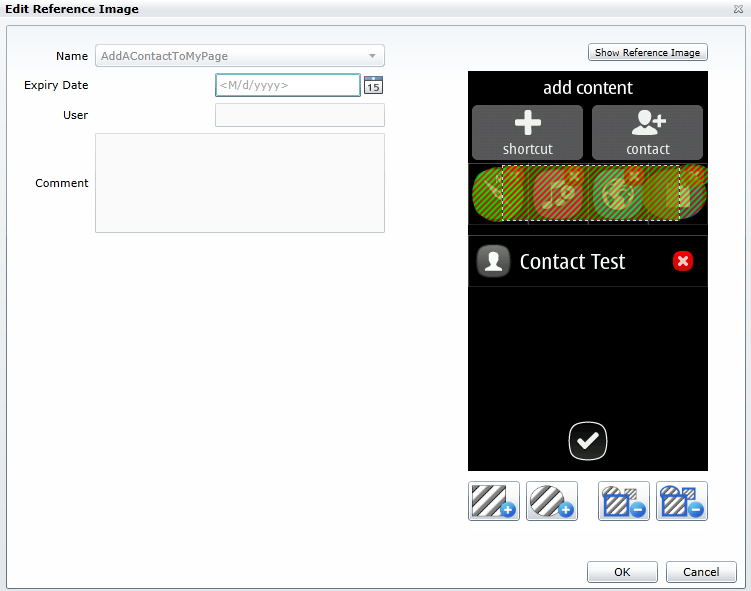
**Option C – XML Mask Meta-Data stored inside image (Accepted)**XML Mask Meta-Data

GUI writes

Activity / framework reads

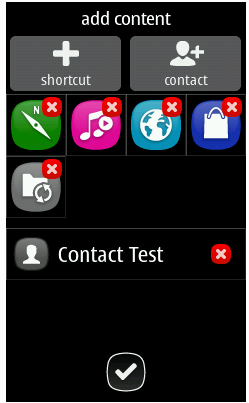
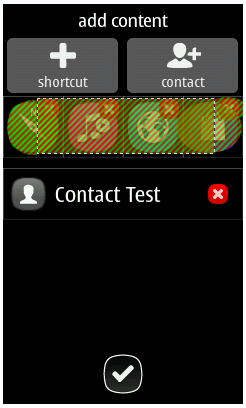
GUI reads

GUI reads



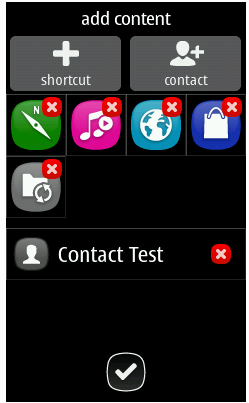
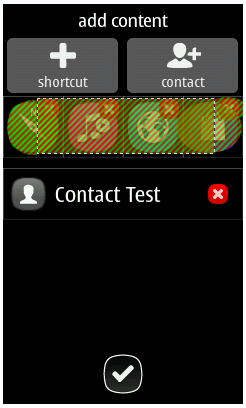
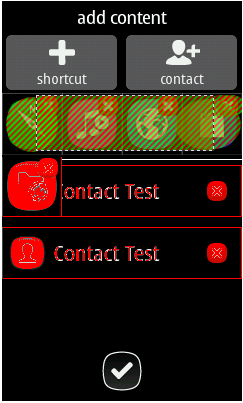
1. User adds reference image and mask

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

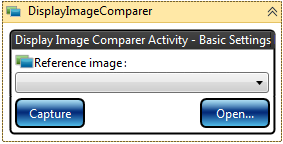
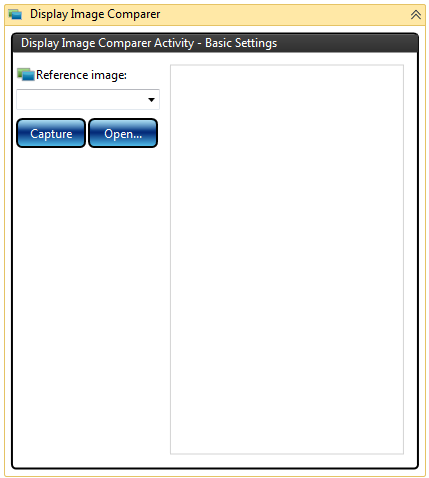
One possible solution could have been to store the reference image itself (i.e. Bitmap) as a property on the DisplayImageComparerActivity. This has the advantage that if a user copies the XAML test case to send to another user, it contains any reference images.

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.

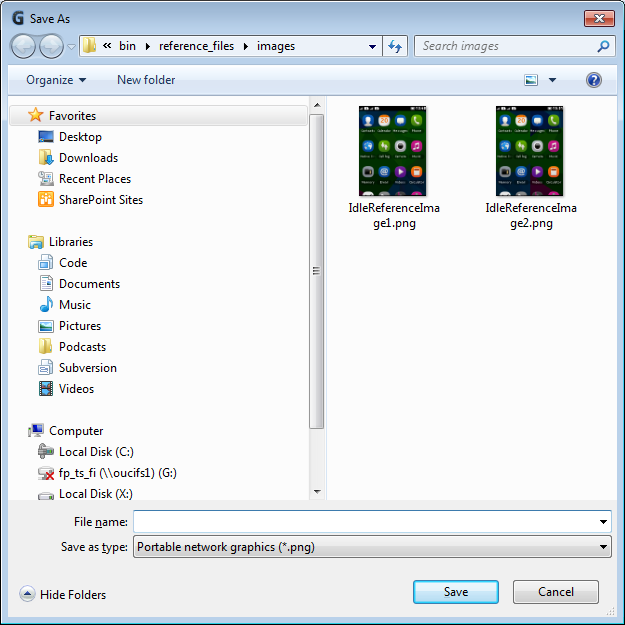
Once we have an image from any of the above sources, we can then display the image in the MaskEditorControl:



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:



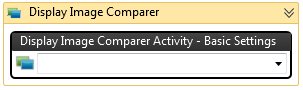
If the user chooses to put the file in a different directory, then the code should display an error message and not save the reference image file.

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**

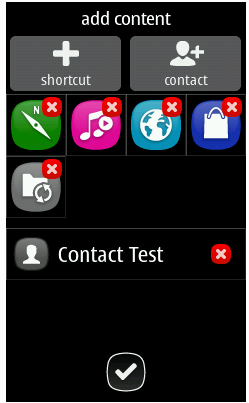
There may be a requirement at a later date to be able to perform reference image comparison from within IronPython scripts. As a result to make this easier, there may be a need to create a tool that helps the developers that write test scripts, to be able to create and maintain reference images using 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.

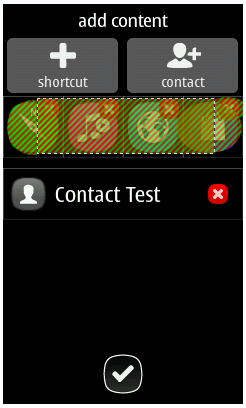
Currently though, the creation and maintenance of the reference images with masks shall only be made available from within Studio. The functionality to perform a comparison with a reference image mask will be available from both Studio and IronPython scripts, since the image comparison functionality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

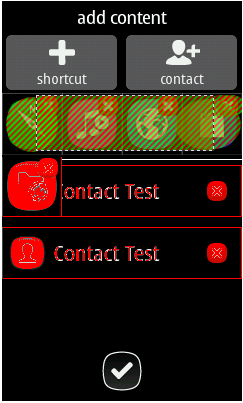
self.compare(referenceImageName)

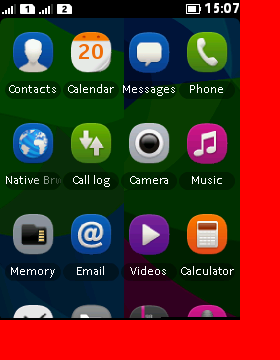
For the report the Framework should generate three images:

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**

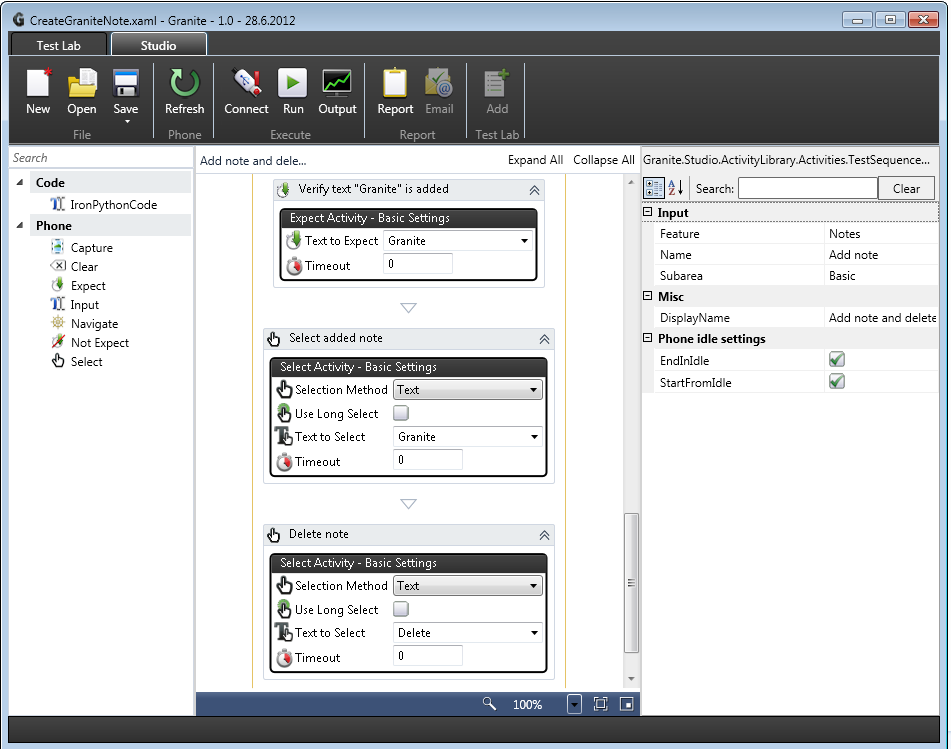


Figure 12 Existing Granite Studio UI and workflow activities

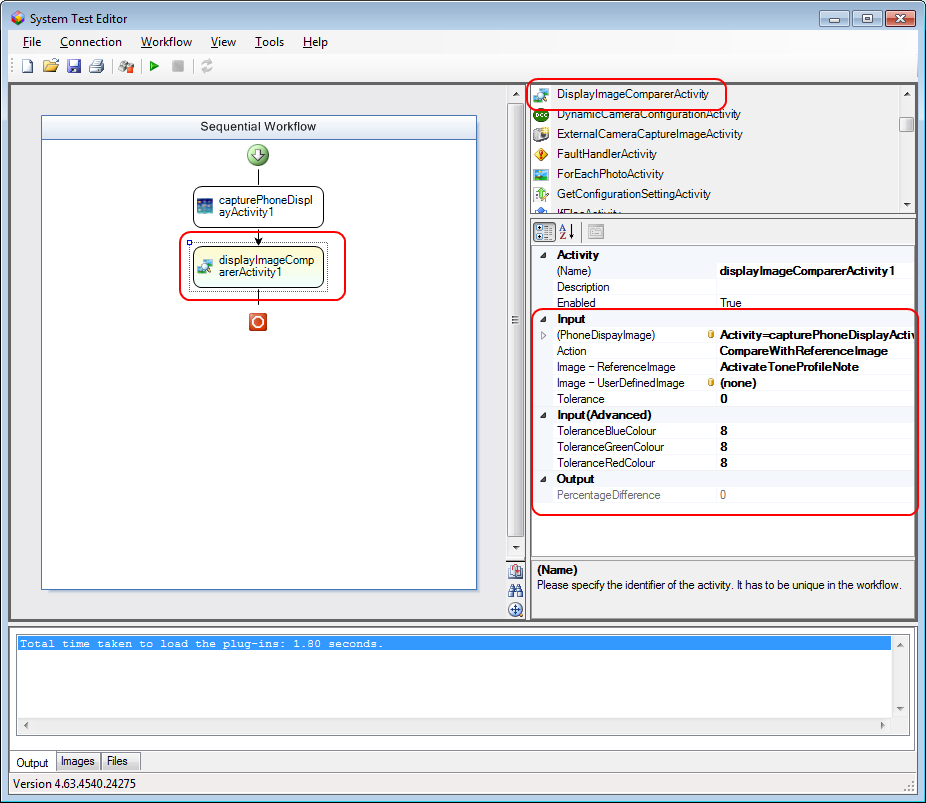


Figure 13 STE with DisplayImageComparerActivity

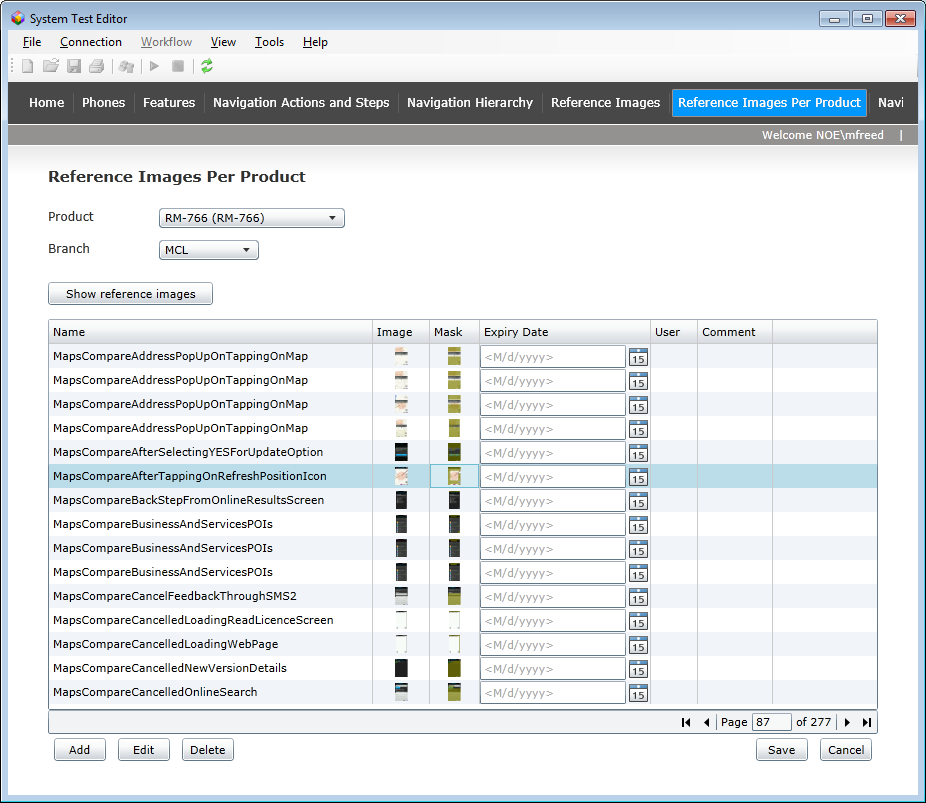
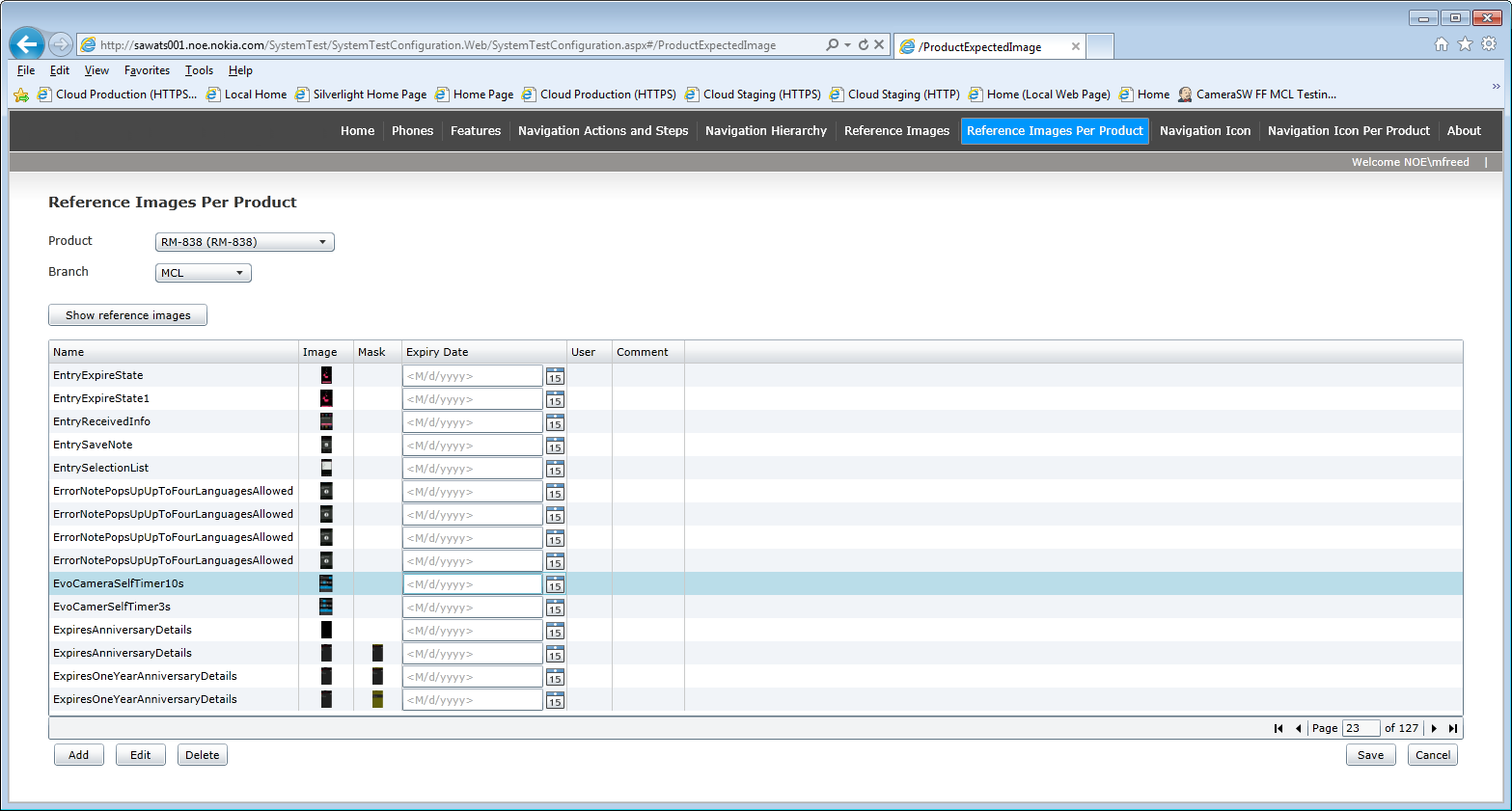
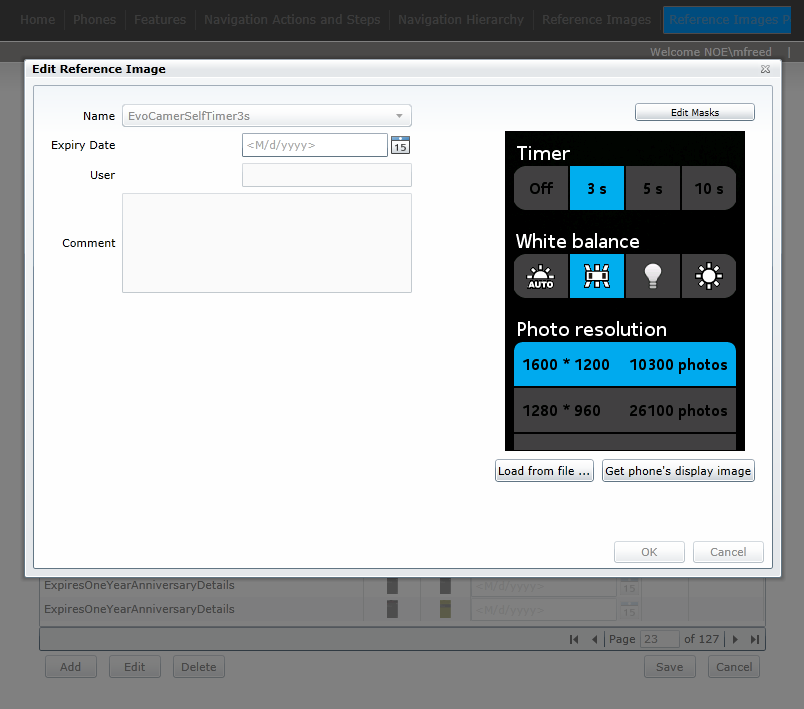
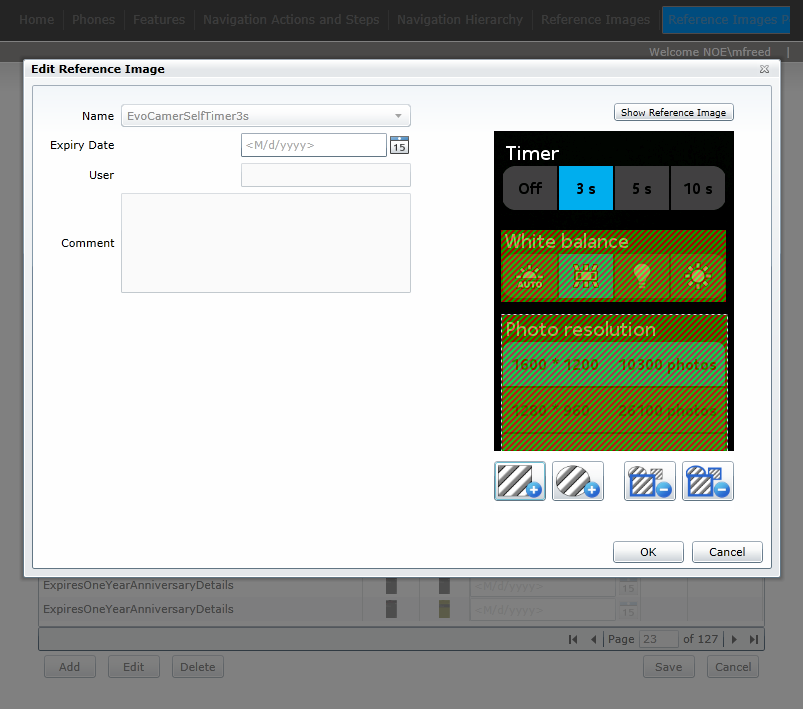
Currently in STE users maintain the reference images and masks through web pages (hosting browser control in STE):

Figure 14 STE with web pages to maintain reference images

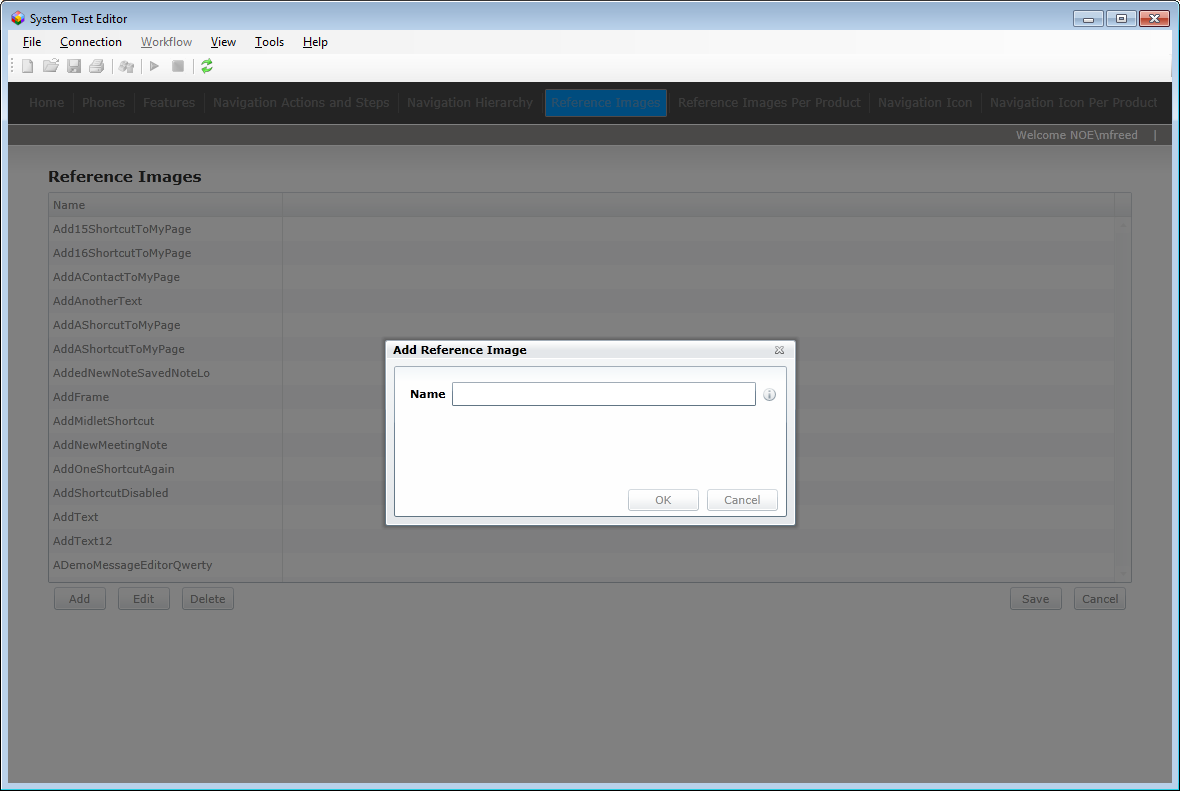
Reference images together with any masks and an expiry date are held in a cloud database. They are partitioned per product, so users can view or maintain reference images on a per product basis.

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

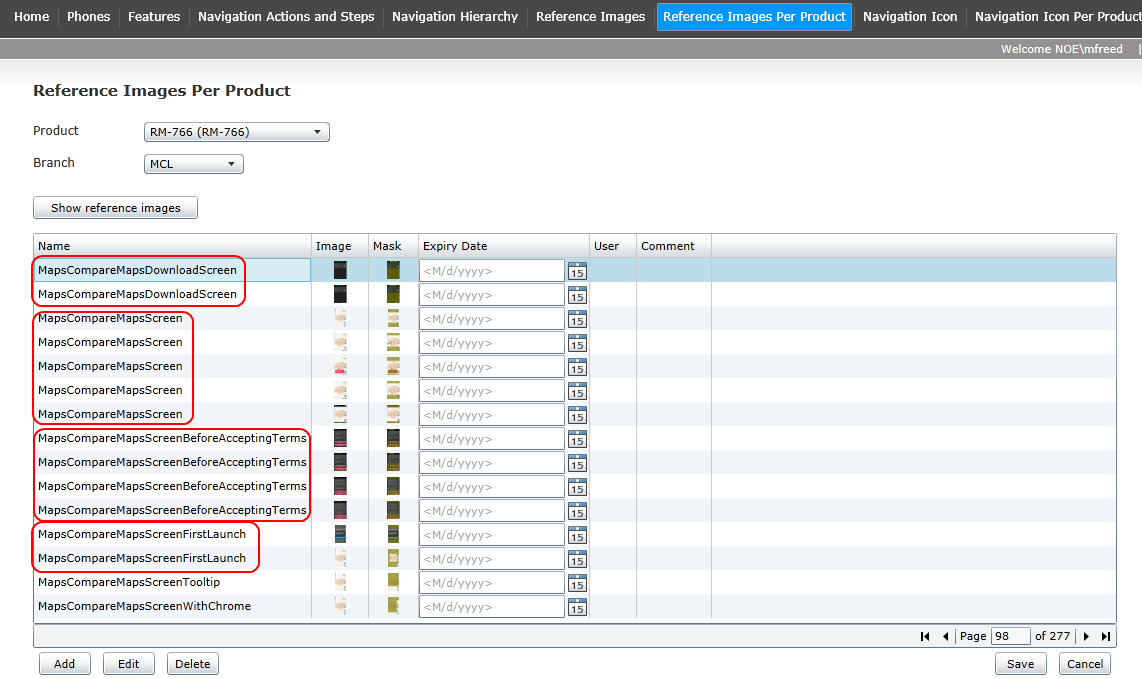
From Granite we need functionality to allow users to add local reference images and masks, similar to the following functionality provided by the STE web pages:



STE users can also add new reference image names:



There is full support in STE for multiple reference images (with the same name) per product:



The image comparison results from a test run can be reviewed from the System Test Image Viewer and the new phone display image can easily be uploaded to the cloud database as a reference image:

