RIM Image Survey Tool

# Development Guide

Laura McCrackin

Last modified: Friday, August 30, 2012

## The RIM Image Survey Tool

The RIM Image Survey Tool is designed for creating, deploying, and aggregating results from image quality surveys. This document briefly describes some of the more technical aspects of the implementation of this tool; for further information, please see this project’s repository on Github: <http://github.rim.net/lmccrackin/Image-Survey-Tool>. If you have any questions, please feel free to contact me at laura.mccrackin@gmail.com.

## Basic Anatomy of the Survey Tool

The Survey Tool contains 2 projects:

* SurveyTool: contains all the main SurveyTool code, and launches the user’s view of the survey by default
* StartSurvey: launches SurveyTool in administrative mode

The following is a brief description of the SurveyTool components:

* StartWindow: the launch window for a user taking the survey
* MainWindow: the window displaying the survey question, which owns the imageviewers
* ImageViewer: a window which displays a single image
* AnswerLoader: the administrative window
* SurveySettings: a class containing all settings from the config file (described later)
* PersonalInfo: contains a person’s name, age, gender, and other personal info
* ImageSet: contains info about a group of images and a list of corresponding Questions
* IQuestions: the interface all Questions must inherit from, which describes all basic logic they must perform
* \_\_\_\_Question: question types that inherit IQuestions
* RadioButtonEnumConverter: minor component, just used for radio button databinding

## Image Set / Question Hierarchy

The Survey Tool is designed to reflect the following assumptions about the layout of a survey:

* Each survey will contain at least one group of images (ImageSet), which are displayed together on-screen and may contain between 1 and 4 images
* Each ImageSet will have at least 1 Question concerning it, with no constraint on the maximum number of Questions
* Questions are displayed one at a time on-screen with their corresponding ImageSet
* Questions may be in different formats (i.e. there are different sub-types of question that may be asked)

## Image Set / Question Hierarchy: Software Design

The hierarchy assumptions stated above were used to design the data structures and software architecture for the Survey Tool. A list of ImageSets is stored in MainWindow.xaml.cs, each of which contains data about the images to display and a list of Questions (which inherit from IQuestions) associated with these images.

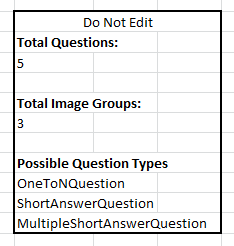
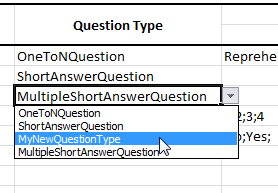
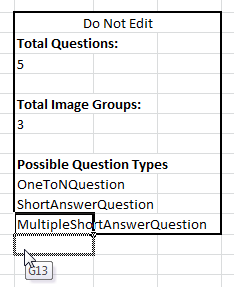
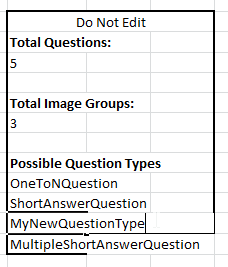
## Question Types

As of this writing, the following Question types (classes extending IQuestions) are available:

* MultipleShortAnswerQuestion
  + One textbox is displayed per image.
* OneToNQuestion
  + For each image, the user is asked to pick a choice from 1 to N, where the choices may be labeled as the survey administrator chooses (ex. Terrible, Bad, Okay, Good, Perfect)
* ShortAnswerQuestion
  + One textbox is provided, regardless of the number of images.

Each Question is responsible for storing its own status information, displaying itself in a Grid that is passed in from MainWindow, reading from and writing its information to .XML during serialization, and aggregating multiple questions of its kind and creating a 2-dimensional string array with

Question types are automatically detected at runtime from the assembly information, so added a new Question type is fairly simple. Just create a new class inheriting both IQuestions and IXMLSerializable, and ensure it implements all required methods, using the other Question types as examples. The new Question type must also be added to the Excel survey template, so that it appears in the drop-down Question Type menu in Excel. Move down the final cell of the Do Not Edit box, and add the new question type in its place. This will add the new type to the drop-down box, as shown below.



## Displaying Images

Between 1 and 4 ImageViewers are created to display images on the screen for any particular ImageSet. These are special windows that load in a bitmap, and allow the user to pan and zoom. An image should be allowed to pan in any direction as long as the current panning change will not expose any less of the current image – this means that the image is not allowed to be scrolled off the canvas and get “lost”. Also note that zooming should be relative to the current centre of the image, as shown in the viewer, and if a zoom action would expose more of the viewer background and less of the image, the image should also be translated accordingly to compensate.

There is a fair bit of math performed to prevent images from being “lost” and to help panning and scrolling behave sensibly, as we have just described. It should be noted that this is not currently working correctly, due to certain properties of the WPF controls in question; see Known Issues for further details.

## The SurveyTool Config File

The SurveyTool uses a config file, config.xml, to store settings. This file resides in the same folder as the SurveyTool executables, and contains the path to the survey Excel file and the path to save finished surveys to.

## Saving a User’s Survey Results

After a user has completed a survey, the survey results are serialized to an xml file, which is named with a random filename in the form “Survey\_randomstringgoeshere.xml” and saved in the folder specified in the survey settings.

To serialize the survey results, the random filename is generated, and a wrapper object, of type SurveyInfoWrapper, is created. This SurveyInfoWrapper contains a reference to the PersonalInfo object containing the user’s personal information, and a list of all ImageSets. When this SurveyInfoWrapper object is serialized, its WriteXML method is executed, which calls the WriteXML method of first the PersonalInfo object, and then for each ImageSet in the list.

The PersonalInfo WriteXML method writes all personal information variables sequentially as elements, and if the user has vision problems, the explanatory textbox string is added as an attribute. In each ImageSet’s WriteXML method, a new element is created for each Question, and this Question’s information is serialized according to its WriteXML method, which is specific to each Question type (subclass).

## Loading in .XML User Survey Results

When a number of user result files have been generated, the survey administrator may then load them in to generate a summary of the results. When this is done, a similar traversal as above is performed for deserializing the data from a result. Each Question type has a CondenseResultsToTable method, which is used to read in a number of the same Question (one for each survey respondent) and aggregate results into a 2-dimensional string array. This grid of text is then written out to an Excel file, which may be saved or manipulated further by the user.

## Known Issues

##### Bugs

* Excel is often not closed tidily by the SurveyTool if the tool is closed prematurely. Perhaps the file handle should be opened in a non-local variable, then disposed of in the OnClose event
* Image panning/zooming does not work correctly. This is because the WPF control values used to calculate this do not quite behave as it seems like they should… when it comes time to fix this, be sure to uncomment the code concerning printing debug vales to the form, as this will be helpful to decode the behavior. Also consider constraining panning to one direction for now.

##### Limitations

* Paths specified to image folders in the Excel survey file must be relative paths to the survey file. This should probably be changed so the user has the option to use an absolute path, too.
* Images in the same ImageSet that are displayed on the screen together should pan/zoom together whenever a user changes the zooming or panning on one of them. This can be done by raising events from each ImageViewer when it is zoomed or panned by the user, and having the MainForm receive these events and pass the new panning and zooming information to its other ImageViewer children.
* There is currently no mechanism in place to check that the Excel file for the survey matches the survey that generated the .xml survey result files. A mismatch will probably cause serious errors.

##### Next Steps

* It has been decided that in the interests of personal privacy, the survey should not ask for information such as the participant’s name. As it was originally proposed that the same set of personal information would always be used for every survey, these fields were hard-coded in. In light of these new considerations, however, it would be better to have the list of personal information questions be flexibly read in, just as the rest of the survey is, from an Excel file or something similar.

##### Future Developments

* Display the current zoom level of photos on the screen (100%, 50%, 200%, etc.)
* Display overlays on images -- A, B, C, D – to help with matching each image with its corresponding question.
* Each question may have a Help / What is this? button, where an additional window is displayed with an example of what the question is asking (ex. for “Rank the noise performance of this image from Good to Bad”, may display examples of “good” and “bad” noise performance in a scene with a brief explanation)
  + Related: Automated training session for survey user before survey questions begin (“This is an image with low noise.  This is an image with high noise”; etc)
* Allow the survey creator to zoom into a region of the image to show the user (initially this will be done in editing the photos in another program and then loaded in the saved files)
* Option for the survey creator to draw a box around an area of the image indicating a region of interest
* Choose image metadata/conditions to present to the person taking the survey (ex. lux level, light type, exposure time, gain – manually entered or from exif data)
* Auto-zooming using Matlab-based image correlation to identically crop images of different sizes (ex. 3MP vs. 5MP)
  + Related: “Smart zooming”, where zooming into one image zooms all others into the same area, even if the camera position is different between them and the subjects are not well-aligned (using correlation of image features to pick the same region from other photos)
* “Tag cloud” question type: user can click on words from a group of keywords (ex. “bright”, “noisy”, “good”, “greenish”) to describe a particular image
  + Related: Could also use this style to summarize data from “short answer” user responses, enlarging/bolding most common words
* Test user’s vision before test – eye chart, colour blindness test
* Tracking which areas of an image a user zooms into, pans across – can use to see what the user looked at
* Remember where windows are placed, possibly across multiple screens, throughout the survey