# Autonomous Cryogenics Loading Operations Simulation Software: Knowledgebase Autonomous Test Engineer

Walter S. Wehner Jr.

KENNEDY SPACE CENTER

Major: Computer Science

Program KSC FO Summer Session

Date: 12 08 2013

# Autonomous Cryogenics Loading Operations Simulation Software: Knowledgebase Autonomous Test Engineer

Walter S. Wehner, Jr.[1]
*New Jersey Institute of Technology, Newark, New Jersey, 07102*

**Working on the ACLO (Autonomous Cryogenics Loading Operations) project I have had the opportunity to add functionality to the physics simulation software known as KATE (Knowledgebase Autonomous Test Engineer), create a new application allowing WYSIWYG (what-you-see-is-what-you-get) creation of KATE schematic files and begin a preliminary design and implementation of a new subsystem that will provide vision services on the IHM (Integrated Health Management) bus. The functionality I added to KATE over the past few months includes a dynamic visual representation of the fluid height in a pipe based on number of gallons of fluid in the pipe and implementing the IHM bus connection within KATE. I also fixed a broken feature in the system called the Browser Display, implemented many bug fixes and made changes to the GUI (Graphical User Interface).**

## Nomenclature

| | | |
|---|---|---|
| $s$ | = | arc length |
| $h$ | = | fluid height |
| $c$ | = | cord length |
| $d$ | = | sector height |
| $R$ | = | pipe radius |
| $\Theta$ | = | sector angle |

## I. Introduction

KATE (Knowledgebase Autonomous Test Engineer) is a software system that uses a physics model of an environment to monitor and identify suspected component failures. The environments are defined by a knowledgebase approach which allows for the abstraction or rules and behaviors from the environment. This level of abstraction gives the knowledgebase approach to modeling environments power and flexibility not available in an expert system. The physics model is implemented in KATE by creating reusable objects. These objects are then consumable in a new knowledgebase. This approach of implementing reusable objects allows a new environment to be modeled starting with all the previous objects.

Working on the KATE codebase I added a GUI (Graphical User Interface) object to KATE that represents the amount of fluid in a horizontal pipe, fixed a broken feature known as the CUI (Compact Unique Identifier) Browser and implemented the connection to the IHM (Infrastructure Health Management) bus. In support of KATE I created an application that allows a schematic to be designed in a WYSIWIG (what-you-see-is-what-you-get) style editor. I also began the design and implementation of a new service that will be available on the IHM bus. This service will provide image analysis and computer vision support to the ACLO (Autonomous Cryogenics Loading Operations) project and the other subsystems on the IHM bus.

The remainder of this paper is organized as follows. Section II will describe the changes and additions to KATE. Section three will describe the xyFile format, the class architecture and the implementation of the Schematic Designer. Section IV will describe the application architecture the class design for camera control, the class design for image analysis and the GUI implementation of the Visual Information Service and Section V will provide conclusions.

[1] Computer Science Masters Student, Computer Science, New Jersey Institute of Technology.

## II. KATE Changes and Additions

The initial KATE project was started in the nineteen nineties as a port from a LISP version of the software to a C++ version. My first few changes to KATE gave me the opportunity to get familiar with the one-hundred plus file codebase and the coding style used by the team that implemented the port. My first changes include added coloring to different display elements so you could easily differentiate between modeled values and physical value; I cleaned up some compiler warnings and fixed some memory leaks that where reported running a tool named valgrind; and I fixed some broken code that caused the CUI Browser to no longer be displayed. Figure one shows the CUI browser and the coloring changes implemented where white backgrounds denote physical elements and blue backgrounds denote modeled elements.

| | | | |
|---|---|---|---|
| GLOX2012E | ---------- | OFF | XFER LINE FILL VLV CLOSED #1 MEAS |
| GLOX3012E | ---------- | OFF | XFER LINE FILL VLV CLOSED #2 MEAS |
| GLOX2013E | ---------- | ON | XFER LINE FILL VLV OPEN #1 MEAS |
| GLOX3013E | ---------- | ON | XFER LINE FILL VLV OPEN #2 MEAS |
| GLOX3033E | ---------- | OFF | XFER LINE FILL VLV SEC SELECTED ME |
| GLOQ0229A | ---------- | 1399.48 | PUMP DISCHARGE FLOW RATE #1 |
| GLOP0226A | ---------- | 189.41 | PUMP DISCHARGE PRESSURE #1 |
| A86461 | | OPEN | XFER LINE FILL VALVE |
| A128 | | OPEN | PUMP A126 CHILL-DOWN VLV (A128) |
| A129 | | CLOSED | PUMP A127 CHILL-DOWN VLV (A129) |

**Figure 1. CUI Browser.**

After familiarizing myself with the codebase and how the system worked internally, I implemented a new feature known as the Fluid Height Gauge control. This new control displays a graphical representation of the fluid height in a horizontal pipe. The requirements for implementing the new control were that it needed to be scalable and the only inputs supplied are pipe diameter, length and number of gallons in the pipe. The control needed to subscribe to updates from the system to provide real-time updates to the number of gallons in a pipe and update its display. I inherited from the class "UpdateObject" and implemented the necessary functions for calculating the fluid height and creating the graphical representation of the pipe cross-section.

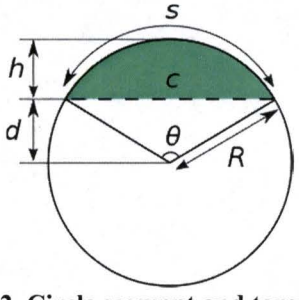



**Figure 2. Circle segment and target area.**

Knowing the length of a pipe and the number of gallons in the pipe would allow me to solve for the diameter of a pipe that would be completely full. Knowing this diameter would then allow me to calculate the cross-sectional area of the volume. This is the target arc area in the original pipe that is full represented in figure 2 as ***c***. Once we calculated the target area to fill in the original pipe we are left with two unknowns in calculating the arc area, the height, ***h***, and an angle theta, Θ.

In figure 3, we present the algorithm for calculating the arc area using an iterative approach incrementing theta and calculating the arc area to find the fluid height.

The process is the same for finding the height above half full but instead of solving for the remaining full we solve for the amount empty and subtract the

```
//////////////////////////////////////////////////////////////
//
//  UpdateFieldFluidHeightGauge::CalculateTheta
//
//  Calculate the angle needed to draw the arc that displays the fluid or empty section
//  Inputs:   radius          - the radius of the pipe
//            targetArcArea   - the area under the sector that would be full or empty
//
//  Outputs:  returns the angle that best fits the area of the sector
//
double UpdateFieldFluidHeightGauge::CalculateTheta(double radius, double targetArcArea)
{
   double increment = .001; //increment of theta when fitting to area
   double theta = 0.0;
   double cordLength = 0.0;
   double arcArea = 0.0;
   double adjustmentTriangle = 0.0;
   double sectorArea = 0.0;
   double beta = 0.0;

   //find the area that represents the fluid
   while (arcArea <= targetArcArea)
   {
     theta += increment;

     cordLength = 2.0 * radius * sin(.5 * theta);

     beta = cos(.5 * theta) * radius;

     adjustmentTriangle = .5 * cordLength * beta;

     sectorArea = ((.5 * theta) * (radius*radius));

     arcArea = sectorArea - adjustmentTriangle;
   }

   return (theta - increment);
}
```

**Figure 3. Algorithm for calculating theta.**

calculated height from the diameter of the pipe. The process of drawing the control also uses the calculated values of theta and fluid height. We convert the values based on the control size into pixels per inch to maintain scaling of different sized controls and various sized pipe segment. In figure 4, we show two different fluid height controls being displayed on a schematic view within KATE.

One requirement for KATE was to be able to communicate with the other systems in the ACLO project. The communication between different systems used an interface called ICE (Internet Communications Engine). I worked with Kelvin Ruiz to integrate the ICE code and implement the necessary changes in KATE for connecting to the IHM (Infrastructure Health Management) bus. Kelvin was designing the GUI interface for what is known to KATE as a Data Provider. The IHM bus was a generic Data Provider class that would take messages and convert them to a KATE friendly structure for consumption within KATE. It also takes messages from KATE and publish these messages for other systems to consume and act upon. I worked on understanding the changes needed for the communications and implementing those changes in KATE while Kelvin completed the GUI. Once Kelvin completed the GUI I worked with him to implement my changes in the new generic data provider module.

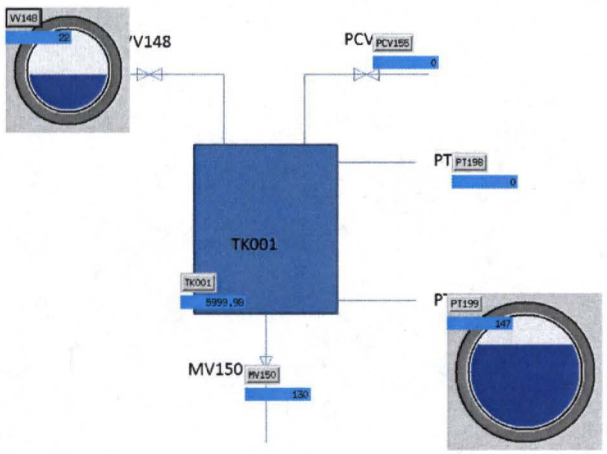



**Figure 4. Fluid Height Gauge GUI Control.**

## III. Schematic xyFile Designer

KATE has a feature that allows for an image to be loaded and then a set of CUI's with values be displayed on top of the image. This feature is known as the Schematic View within the KATE application. The current method of developing these files is by opening your favorite text editor and hand coding all the values. The Schematic xyFile Designer application developed is a utility program created for windows that allows for the generation of "schematic xy files" for use in KATE. This application allows for the WYSIWYG creation of the xy files so a user does not need to know the internal structure to create the file or hand code the file in a text editor.

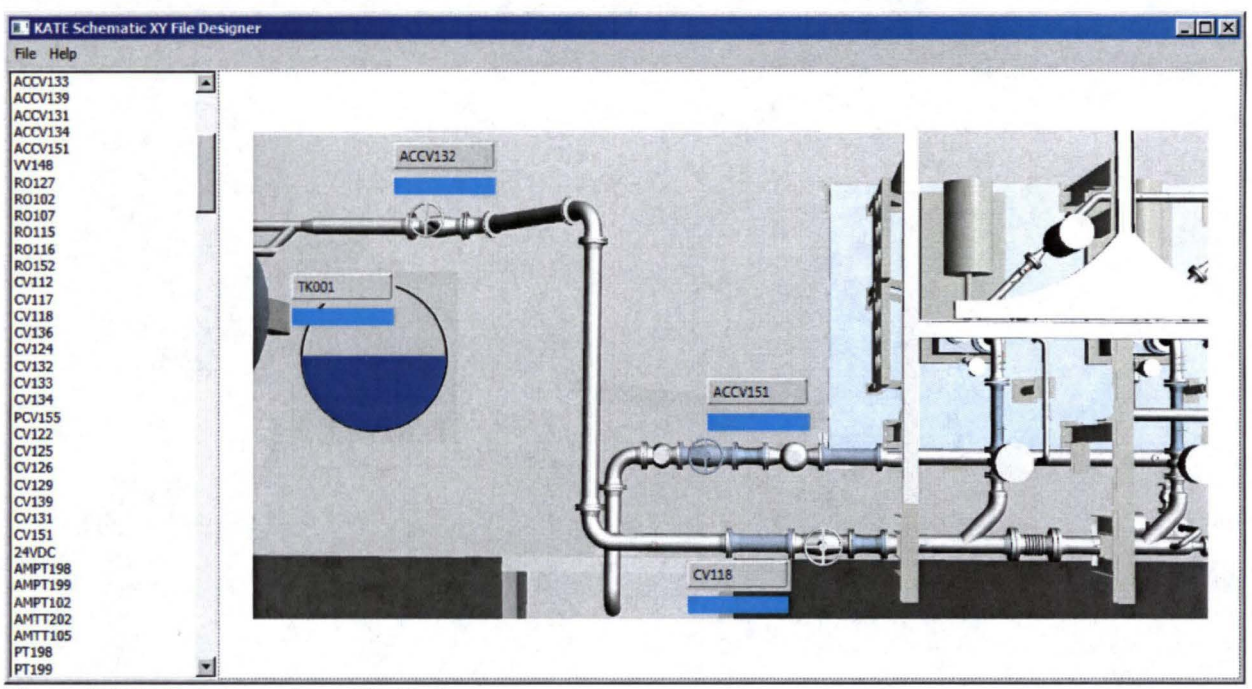



**Figure 5. KATE Schematic xy File Designer.**

### A. KATE xy File Format

The KATE xy file has a specific structure that allows for the loading of the graphical objects to be displayed on the schematic view. The encoding of the file is ASCII and the newline character is '\n' as opposed to '\r\n' which is standard in windows. The first line of the file is XYFILE followed by a new line. After the first line, the rest of the file is a listing of the elements to be displayed which are separated by a newline. The element information block consists of up to fifteen parameters depending on the class type being displayed. The first parameter is the index value. The second parameter is the class type value and will determine if we load additional values.

| Value Display Class (classtype = 0) | Fluid Height Gauge (classtype = 1) | Tank Fluid Height (classtype = 2) |
|---|---|---|
| name (text)<br>x_loc (integer)<br>y_loc (integer)<br>x1 (integer)<br>x2 (interger)<br>show (integer) | pipe diameter (floating point)<br>pipe length (floating point)<br>control width (integer)<br>control height (integer)<br>name (text)<br>x_loc (integer)<br>y_loc (integer)<br>x1 (integer)<br>x2 (interger)<br>show (integer) | tank top height (floating point)<br>tank mid height (floating point)<br>tank bottom height (floating point)<br>semi major axis (floating point)<br>semi minor axis (floating point)<br>control width (integer)<br>control height (integer)<br>name (text)<br>x_loc (integer)<br>y_loc (integer)<br>x1 (integer)<br>x2 (interger)<br>show (integer) |

**Table 1. xy File types and associated parameters.**

Table 1 shows the additional parameters for each classtype loaded. All parameters are delimited by a new line character.

### B. Class Design

The class structure implemented begins with an abstract class called "KnowledgebaseItem". This abstract class in inherited by the "ValueDisplayItem", "FluidHeightGaugeItem" and "TankFluidHeightItem" classes. These classes hold the data that is written to the file and implement the interface to alert the framework when a property value has change. To hold these items and use them we created a classes called "KnowledgebaseItemList" which extends the "ObservableCollection" framework class. This list will notify the framework when there are changes to the collection such as adding and removing elements.

We then must implement the controls to display our data. We start by designing a class called "KnowledgebaseItemVisual" which will be exteneded byt three children called "ValueDisplayItemVisual", "FluidHeightGaugeVisual" and "TankFluidHeightVisual". These classes handle drawing an individual knowledgebase item on the screen. Next, we create a control called the "KnowledgebaseItemListVisualizer". This class extends the class "FrameworkElement" and implements listening to changes in the "KnowledgebaseItemList", drawing the "KnowledgebasedItemVisual" classes.

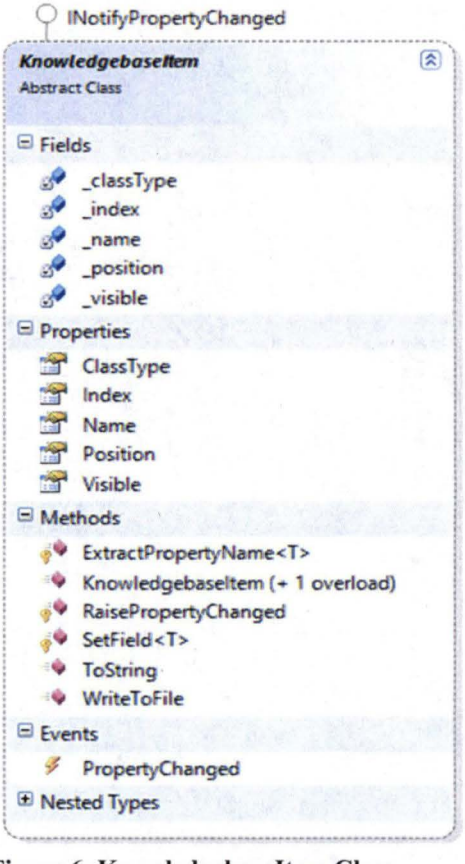



**Figure 6. KnowledgebaseItem Class.**

### C. Implementation

The implementation of our application is done by creating a user control called "SchematicDesigner". The "SchematicDesigner" class which inherits from the class "UserControl". The "SchematicDesigner" is a control that handles adding and removing "KnowledgebaseItemVisual", displaying a

schematic image background and the dragging and dropping of the items into place. In the main application window we use

this control and a ListBox control that loads the KnowledgebaseItems. We are able to drag the CUI name from the ListBox and move them in place on the SchematicDesigner control. The mail window class handles loading a "KATE flatfile", loading a "Kate xy File" and saving a "KATE xy file".

We show the WYSIWYG Schematic Designer application in Figure 5. The application is displaying a Fluid Height Gauge and multiple Display Value items over a background image and a list of CUI's that are able to be added to the schematic.

Please see the Appendix for a review of all class diagrams used in the KATE Schematic xyFile Designer.

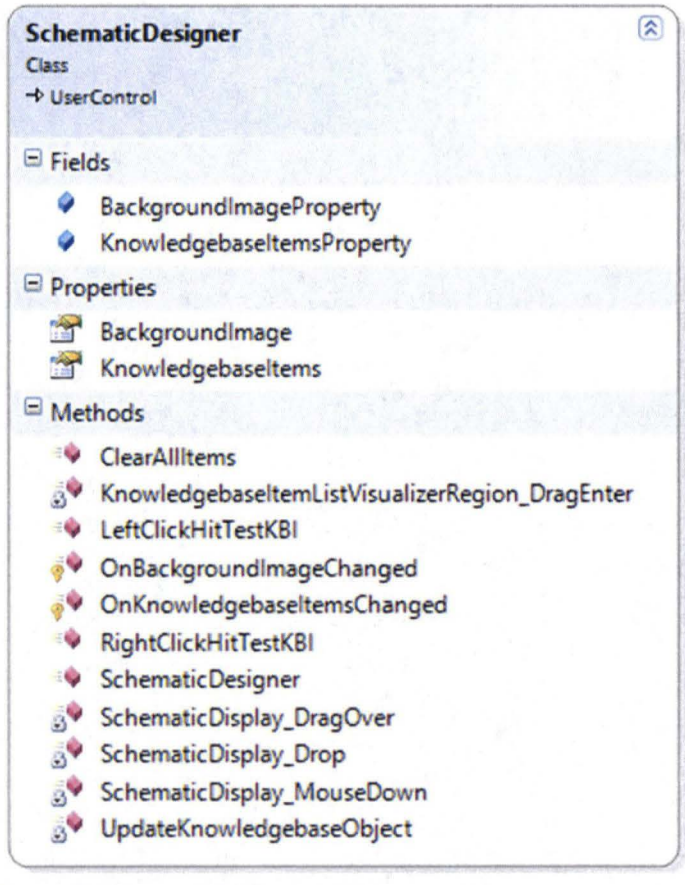



**Figure 7. SchematicDesigner Class.**

## IV. Visual Information Services Application

The VIS (Visual Information Services) is an application that will add computer vision and image analysis support to the ACLO project. The requirements for VIS are that it be a windows based application but the application logic be developed in C++ and be portable to Linux; and the application will be available for other systems on the IHM bus.

### D. Application Architecture

The Windows GUI is written in C# using WPF (Windows Presentation Foundation). C# is a managed language and uses the .NET Framework. The application code is packaged in a DLL (Dynamic Link Library) and is written in native C++. To use the native DLL in the .NET Framework I needed to implement a managed C++ wrapper to handle creating and disposing of the native objects and converting between native data types and managed data types. There are many benefits to this design including portability between platforms; flexibility in GUI design and application implementation; and reusability of image processing and computer vision functions.

I have created a DLL called "CameraControl.dll" that is responsible for communications between the application and the network cameras in the Cryogenics Test Bed. This DLL provides functions to move a camera, retrieve images from a camera and query/set some camera parameters. This DLL is written in native C++ and is wrapped by a DLL called "ApplicationBridge.dll". I expose a few underlying classes in this application wrapper and handle all

memory management for the native object creation. The "ApplicationBridge.dll" is then consumed by the GUI written in C# and I use the GUI to load a database of CUI locations, request images and perform image processing and analysis.

The image processing and analysis routines are in another DLL which is currently written in C#. The goal will be to write this DLL in native C++ to make it portable between platforms and consumable by any client application. In the image processing DLL we have implemented functions for gray scale conversion, binary conversion, skeletonization, edge detection using Sobel filtering, image smoothing and circle detection by Edge Oriented Hough Transform.

### E. Camera Control and Application Bridge Classes

The "CameraControl.dll" contains the classes used to communicate directly to the cameras. We have a class "Camera" that implements all the necessary functions to directly communicate with the camera. There is a class called "CameraClient" that has a camera and implements high level camera access. This allows the consumer to call a single function to retrieve an image or move the camera and the class implements all the calls needed to the camera.

The "ApplicationBridge.dll" DLL provides the managed classes used in C#. We have wrapped the needed "CameraControl.dll" classes and implemented all the native memory management. The "CameraClient" class is wrapped as "ManagedCameraClient" and provides the managed facilities to communicate at a high level with the camera.

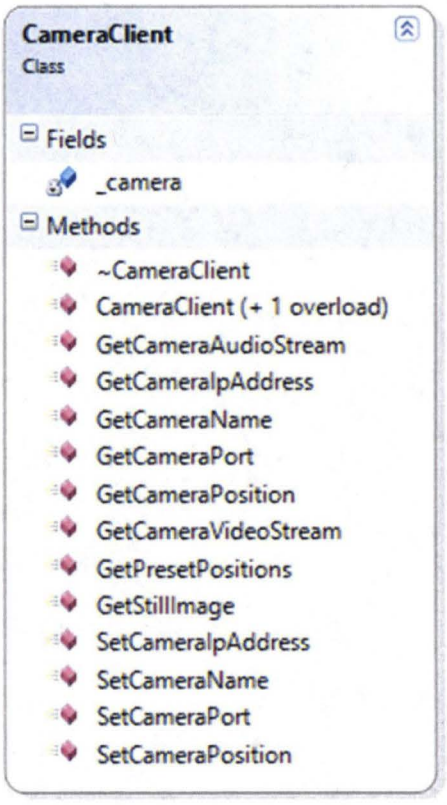



**Figure 8. CameraClient Class.**

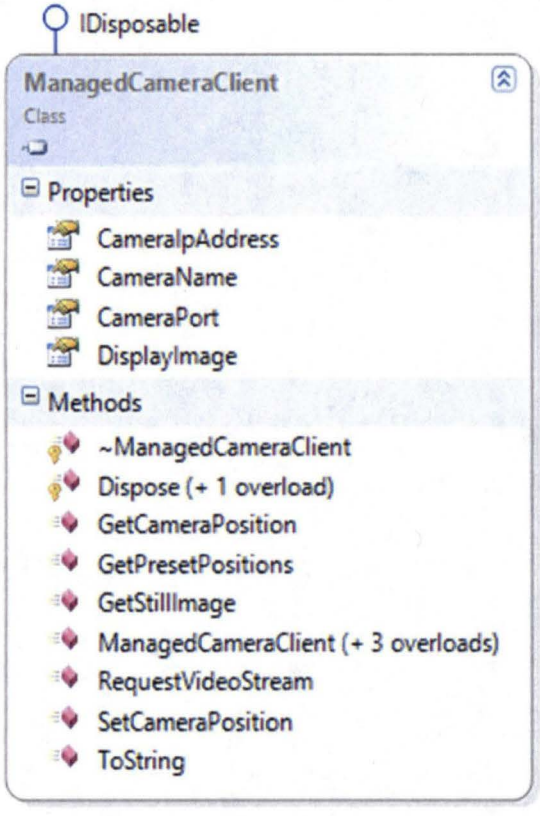



**Figure 9. ManagedCameraClient Class.**

The additional classes implemented in "CameraControl.dll" are "CameraParameter", "CameraPosition", "MPEG4Stream", "AudioStream", "CameraPresetPositionList", and "CameraHttpResponse". Please see the appendix for the class diagrams of all classes in the "CameraControl.dll".

### F. Image Processing and Analysis Classes

The main image processing and analysis classes currently implemented are "ImageUtilites", "HoughCircle", "Cluster", "Skeletonization" and "ImageMatrix". "ImageUtilites" provides functions to manipulate an image such as image binarization, image smoothing, edge detection and thinning. "HoughCircle" provides the facilities for circle detection. "Cluster" is used to group candidate circles together when performing object detection. "Skeletonization" provides the implementation of an image thinning algorithm and "ImageMatrix" is used for directly accessing and manipulating an image and its pixels.

Please see the appendix for the class diagrams in the Image Processing library.

### G. GUI Implementation

The implementation of our GUI is done in C#. The main application provides the facilities to load a database of cameras and their IP addresses, load a database of

predefined CUI's and their locations, request and display an image from a camera, query the camera for its list of preset positions and perform analysis on the images to detect gauges.

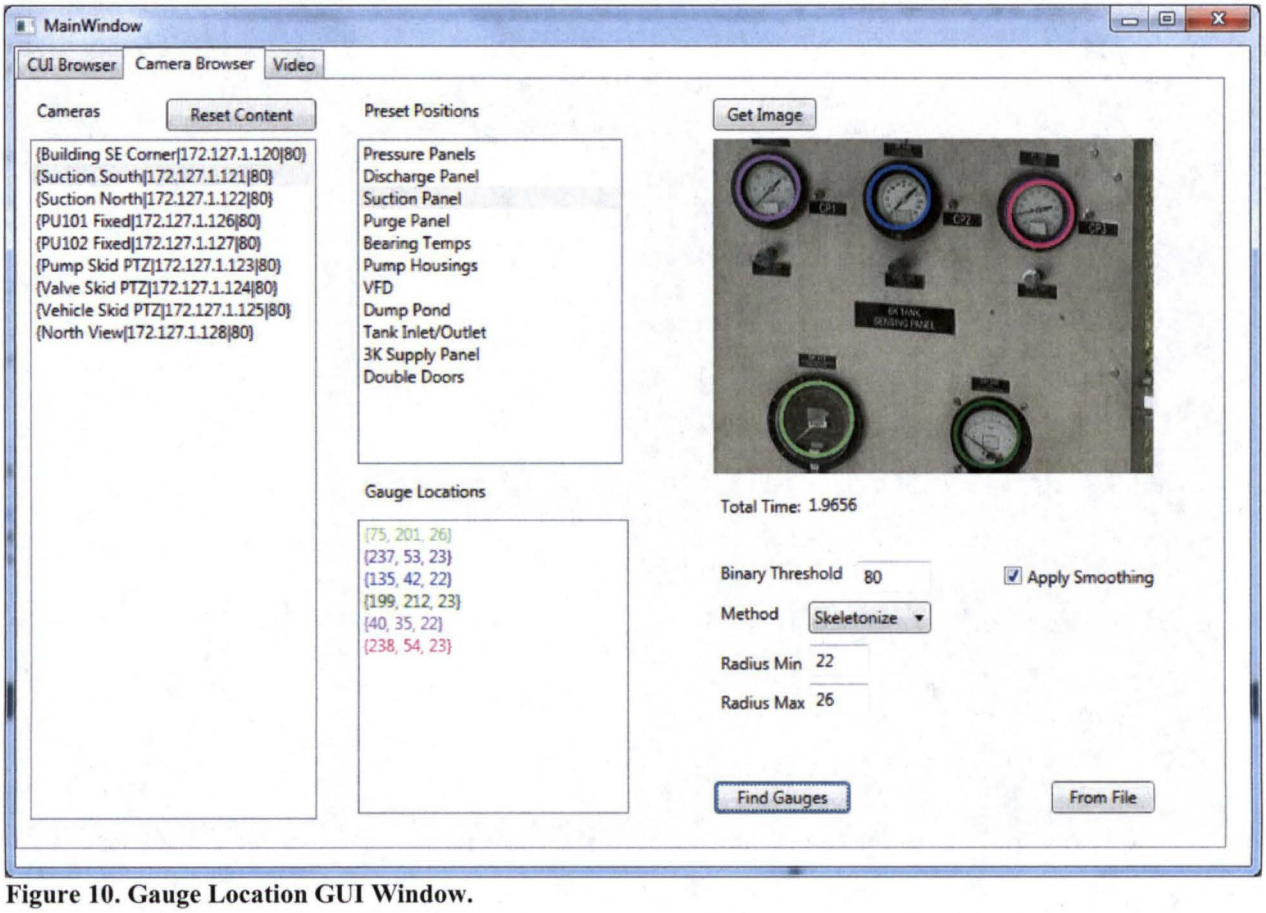



**Figure 10. Gauge Location GUI Window.**

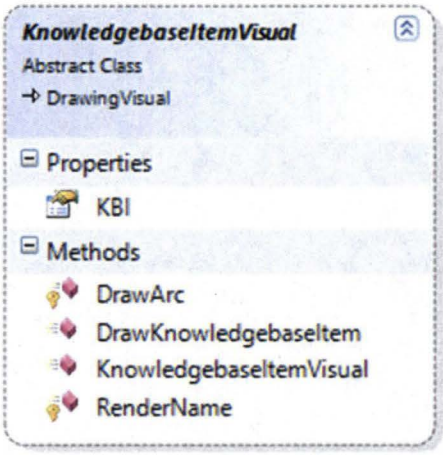
## V. Conclusion

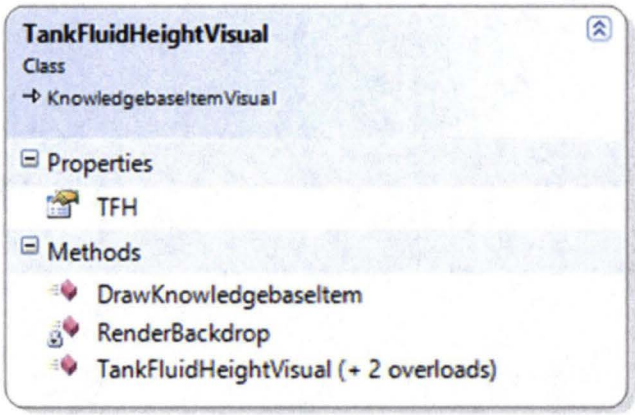
The projects I worked on have been a great learning experience. I see KATE as a great tool for use in the future and the computer vision application I begin developing will be valuable to every system on the IHM bus. I have developed a framework for this new computer vision service and implemented some features to show proof of concept. Some of the services I implemented in the new system are the ability to move cameras to see specific CUI's; being able to take images and send them to clients; and being able to identify different gauges. Some features for future expansion will include reading gauge values; automatically identifying all the CUI's in the system and creating a 3-D map of the CUI's. These features will allow the automatic creation of KATE Knowledgebase files which can contain thousands of items currently entered by hand and would be useful when there are configuration changes. As another data source for KATE it will also enhance the reasoning ability for fault detection and diagnosis of a failed component. Expanding the vision services beyond CUI's and values to inputs such as boiling liquid vapor recognition and infrared heat maps of the flow path would also be very useful data for KATE.

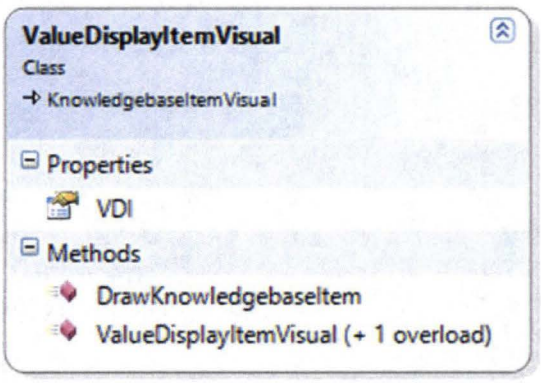
## Appendix

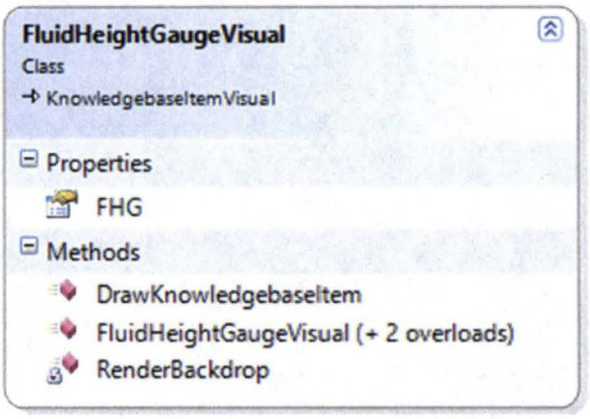
The Appendix is organized into three sub-sections. The classes for the KATE Schematic xyFile Designer are listed in part A; the classes in the CameraControl DLL are shown in part B; in part C we show the classes in the ApplicationBridge DLL and Camera Control GUI; and the classes in the Image Processing DLL are shown in Part D.

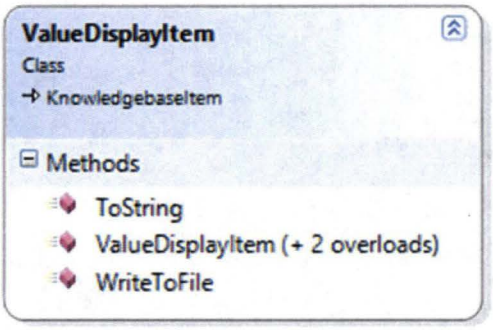
**A. KATE Schematic xyFile Designer Classes**

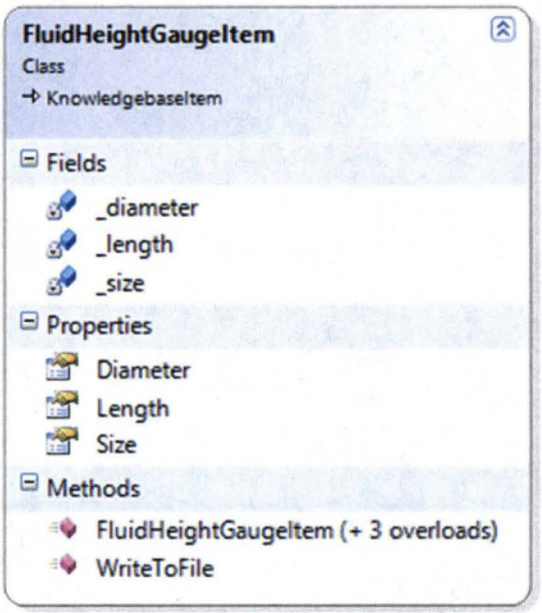


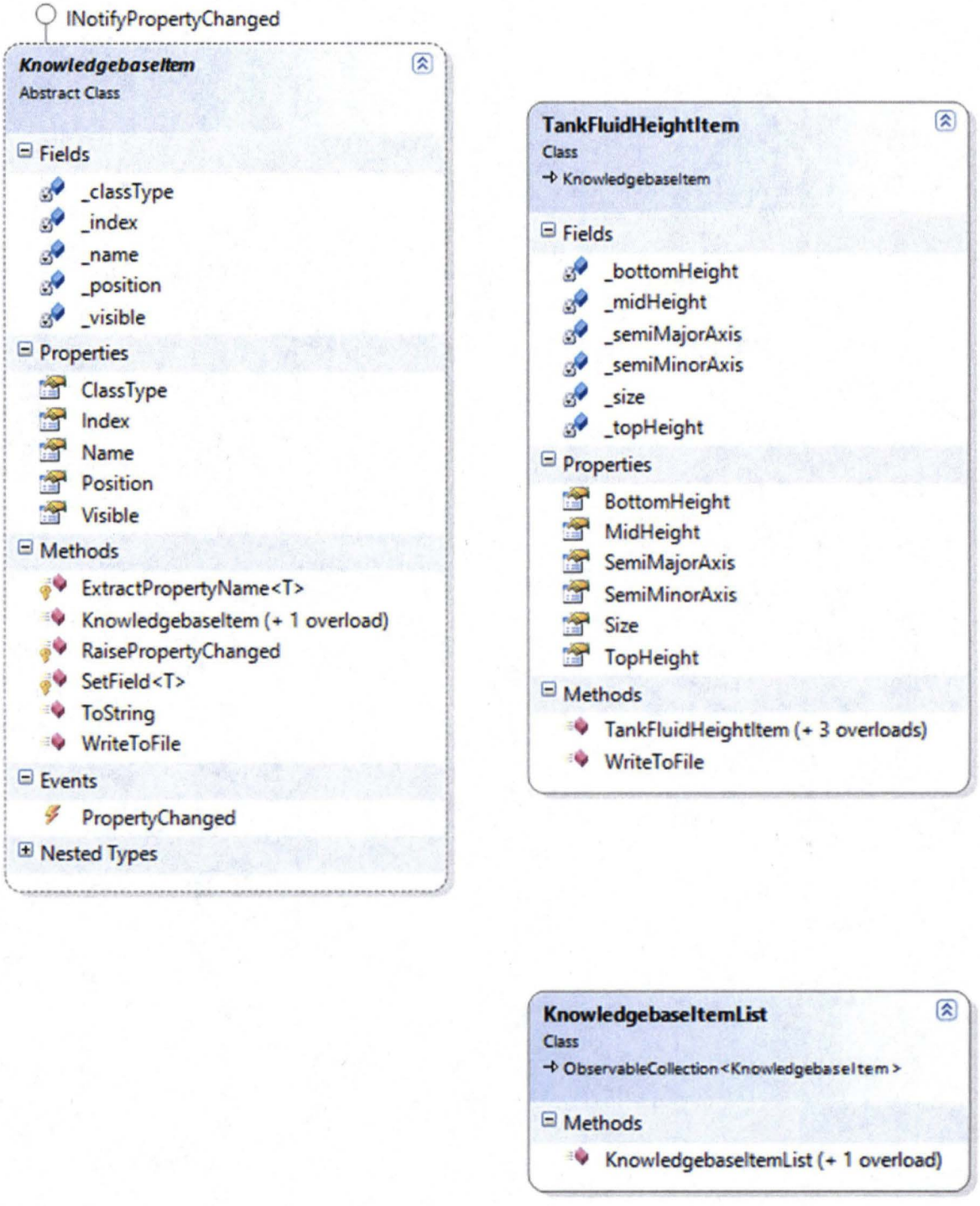



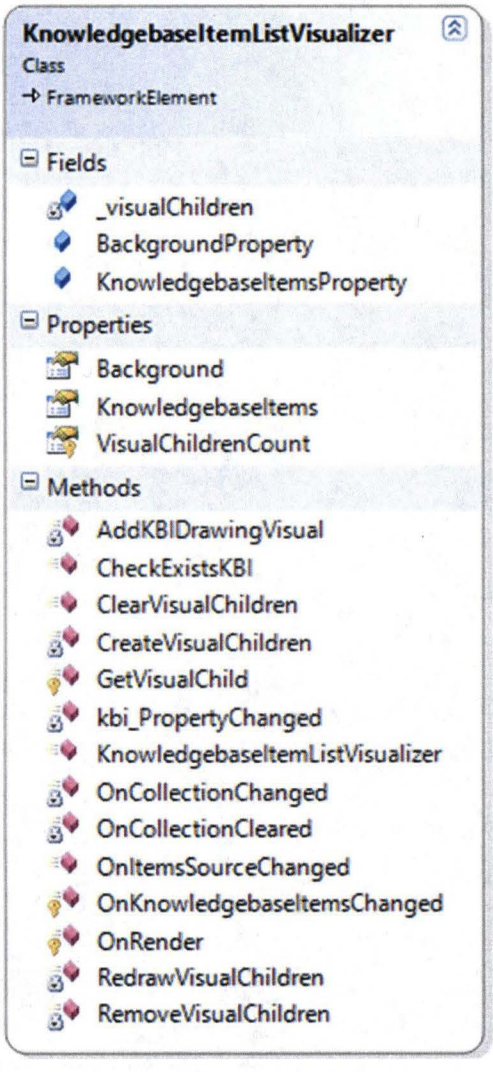



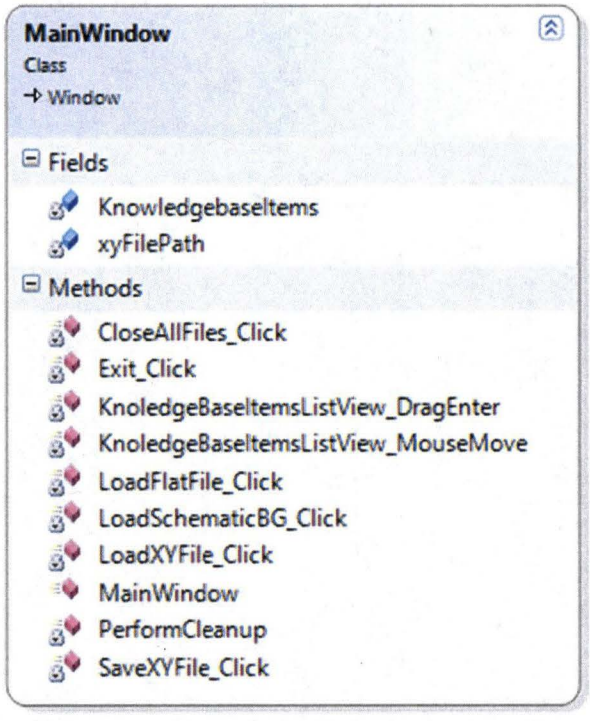



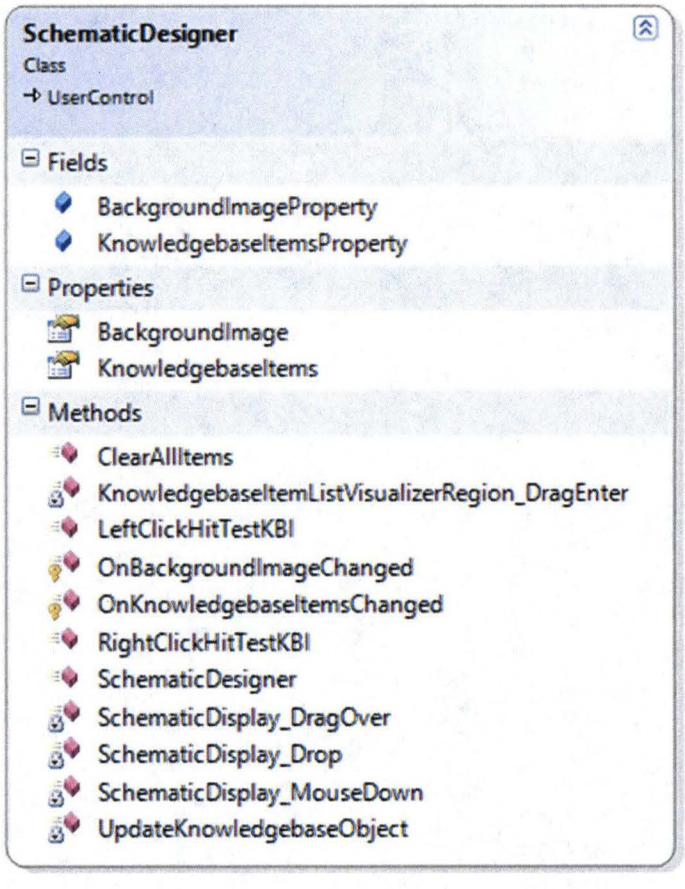



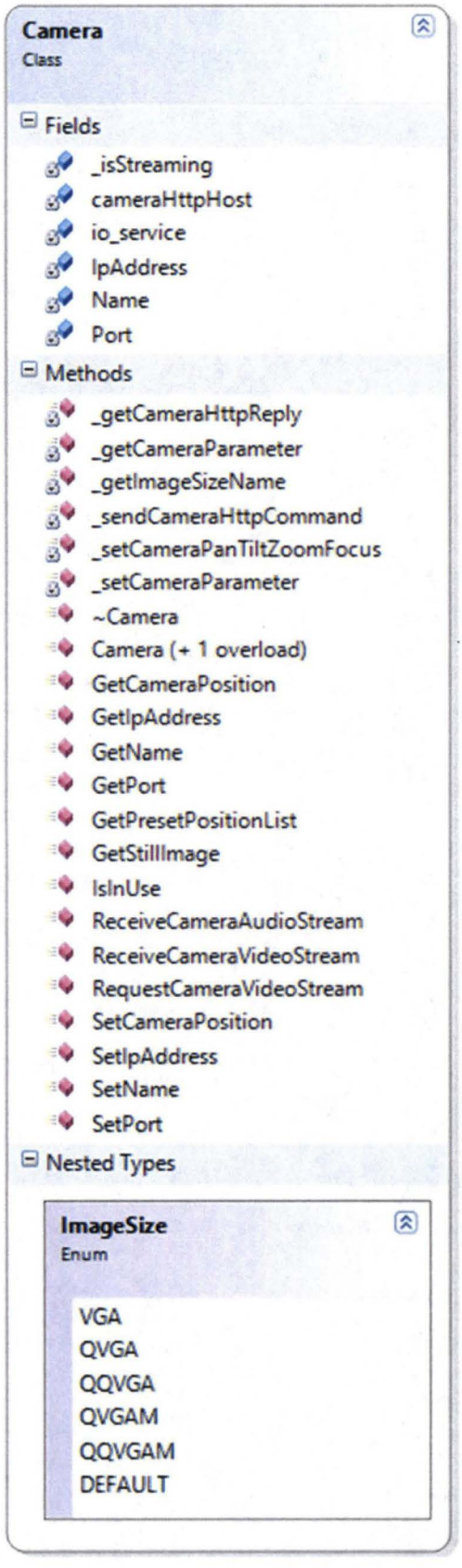

INotifyPropertyChanged
KnowledgebaseItem
Abstract Class
Fields
_classType
_index
_name
_position
_visible
Properties
ClassType
Index
Name
Position
Visible
Methods
ExtractPropertyName<T>
KnowledgebaseItem (+ 1 overload)
RaisePropertyChanged
SetField<T>
ToString
WriteToFile
Events
PropertyChanged
Nested Types
TankFluidHeightItem
Class
KnowledgebaseItem
Fields
_bottomHeight
_midHeight
_semiMajorAxis
_semiMinorAxis
_size
_topHeight
Properties
BottomHeight
MidHeight
SemiMajorAxis
SemiMinorAxis
Size
TopHeight
Methods
TankFluidHeightItem (+ 3 overloads)
WriteToFile
KnowledgebaseItemList
Class
ObservableCollection<KnowledgebaseItem>
Methods
KnowledgebaseItemList (+ 1 overload)

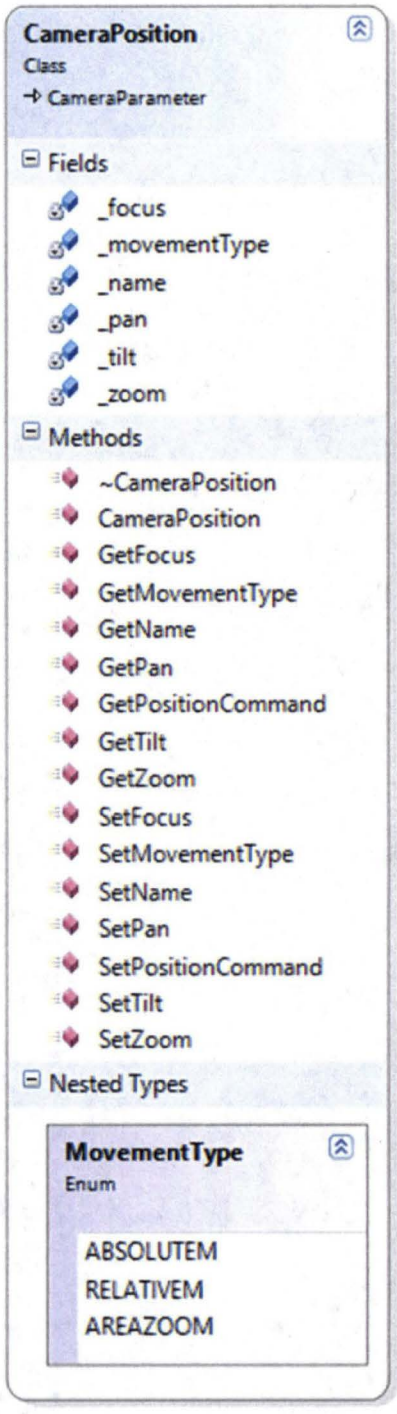

KnowledgebaseItemListVisualizer
Class
FrameworkElement
Fields
_visualChildren
BackgroundProperty
KnowledgebaseItemsProperty
Properties
Background
KnowledgebaseItems
VisualChildrenCount
Methods
AddKBIDrawingVisual
CheckExistsKBI
ClearVisualChildren
CreateVisualChildren
GetVisualChild
kbi_PropertyChanged
KnowledgebaseItemListVisualizer
OnCollectionChanged
OnCollectionCleared
OnItemsSourceChanged
OnKnowledgebaseItemsChanged
OnRender
RedrawVisualChildren
RemoveVisualChildren

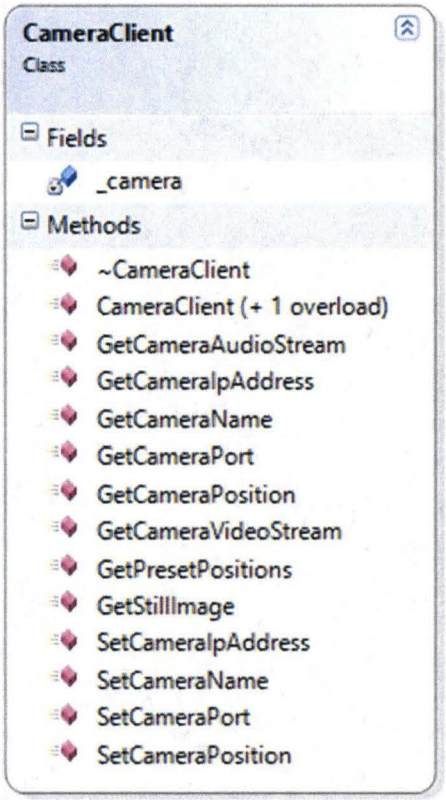

MainWindow
Class
Window
Fields
KnowledgebaseItems
xyFilePath
Methods
CloseAllFiles_Click
Exit_Click
KnoledgeBaseItemsListView_DragEnter
KnoledgeBaseItemsListView_MouseMove
LoadFlatFile_Click
LoadSchematicBG_Click
LoadXYFile_Click
MainWindow
PerformCleanup
SaveXYFile_Click

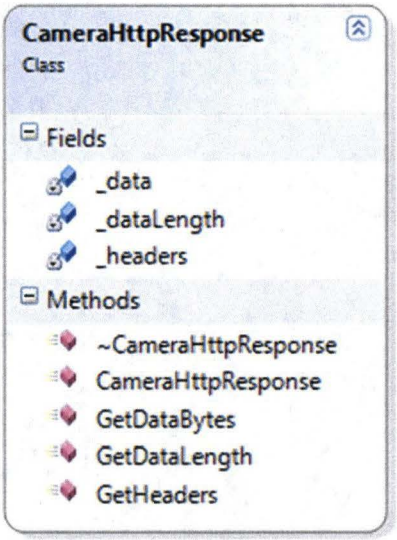

SchematicDesigner
Class
UserControl
Fields
BackgroundImageProperty
KnowledgebaseItemsProperty
Properties
BackgroundImage
KnowledgebaseItems
Methods
ClearAllItems
KnowledgebaseItemListVisualizerRegion_DragEnter
LeftClickHitTestKBI
OnBackgroundImageChanged
OnKnowledgebaseItemsChanged
RightClickHitTestKBI
SchematicDesigner
SchematicDisplay_DragOver
SchematicDisplay_Drop
SchematicDisplay_MouseDown
UpdateKnowledgebaseObject

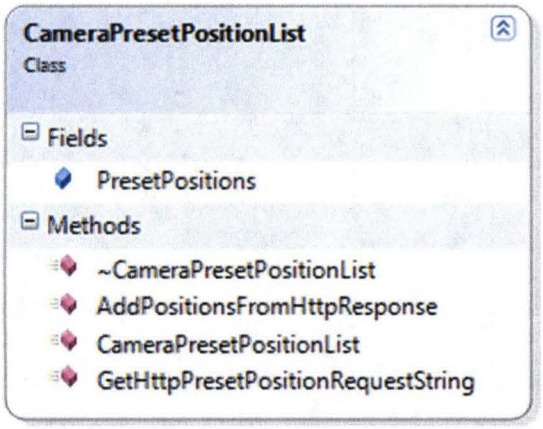
**B. CameraControl DLL Classes**

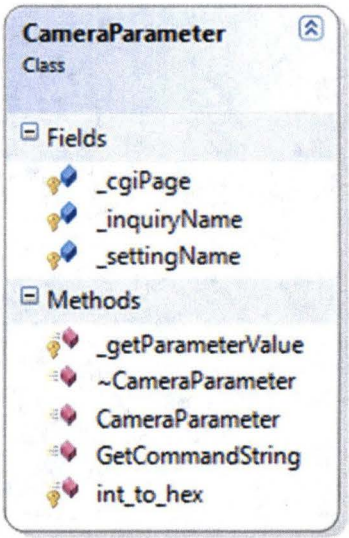


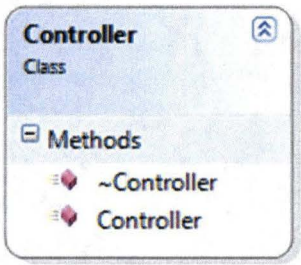




CameraClient
Class
Fields
_camera
Methods
~CameraClient
CameraClient (+ 1 overload)
GetCameraAudioStream
GetCameraIpAddress
GetCameraName
GetCameraPort
GetCameraPosition
GetCameraVideoStream
GetPresetPositions
GetStillImage
SetCameraIpAddress
SetCameraName
SetCameraPort
SetCameraPosition

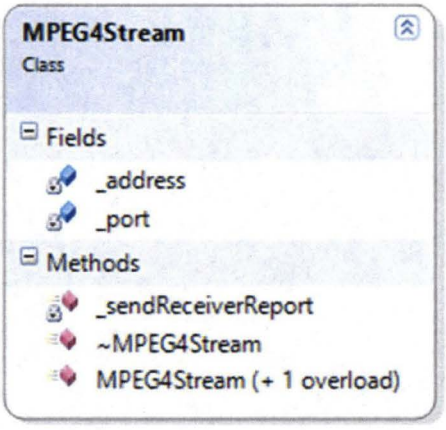

CameraHttpResponse
Class
Fields
_data
_dataLength
_headers
Methods
~CameraHttpResponse
CameraHttpResponse
GetDataBytes
GetDataLength
GetHeaders


CameraPresetPositionList
Class
Fields
PresetPositions
Methods
~CameraPresetPositionList
AddPositionsFromHttpResponse
CameraPresetPositionList
GetHttpPresetPositionRequestString

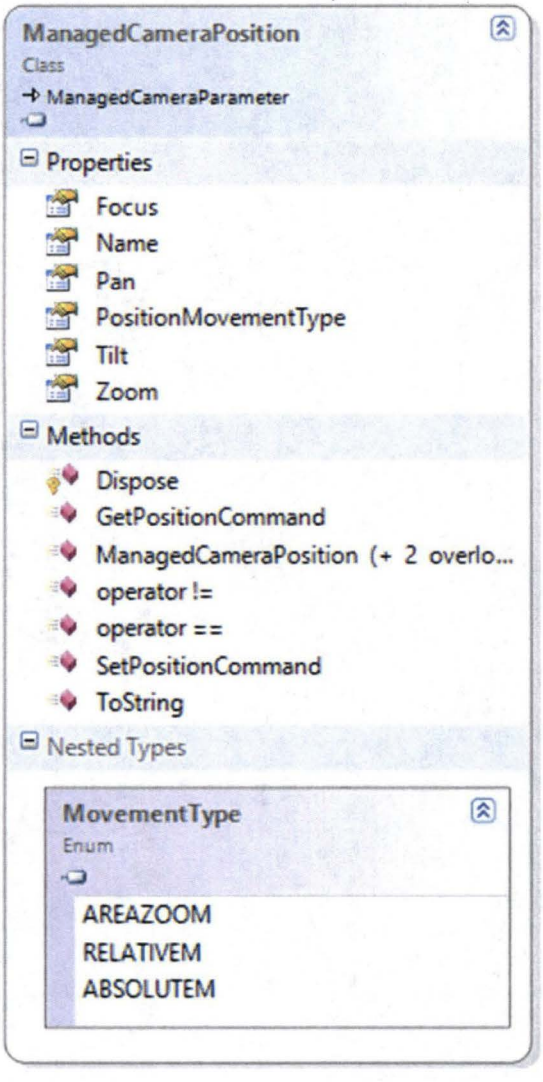

CameraParameter
Class
Fields
_cgiPage
_inquiryName
_settingName
Methods
_getParameterValue
~CameraParameter
CameraParameter
GetCommandString
int_to_hex

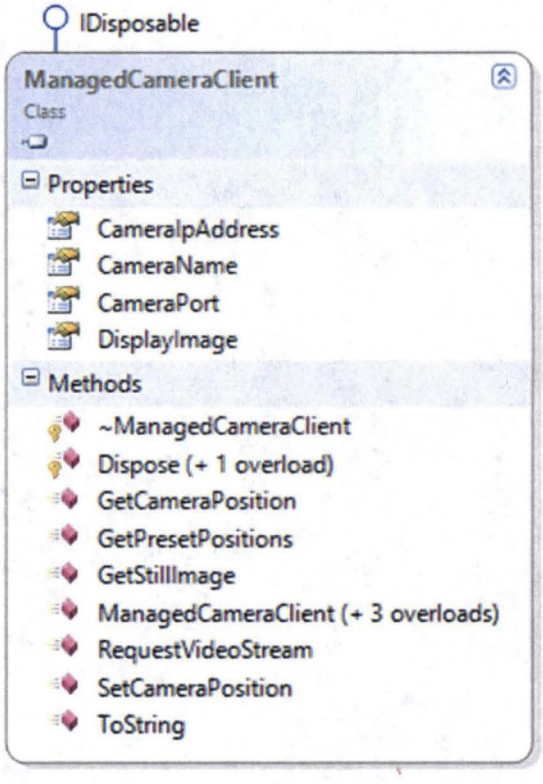

Controller
Class
Methods
~Controller
Controller

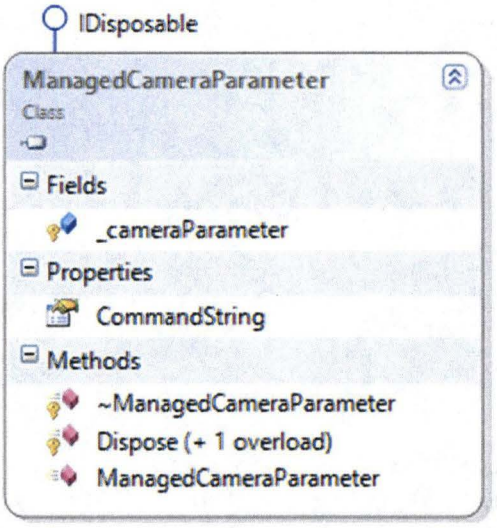

AudioStream
Class
Methods
~AudioStream
AudioStream

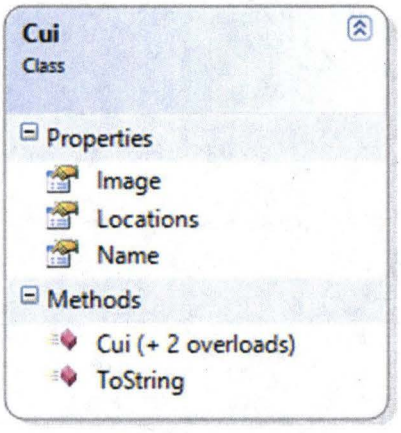


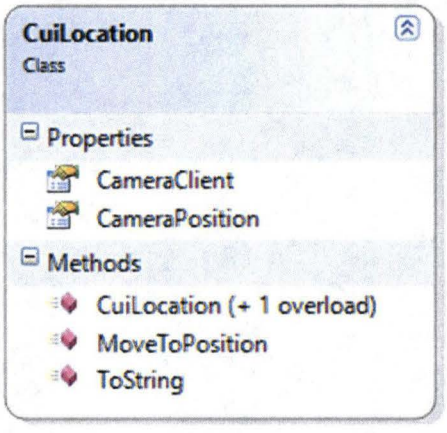

### C. Application Bridge DLL and GUI Classes



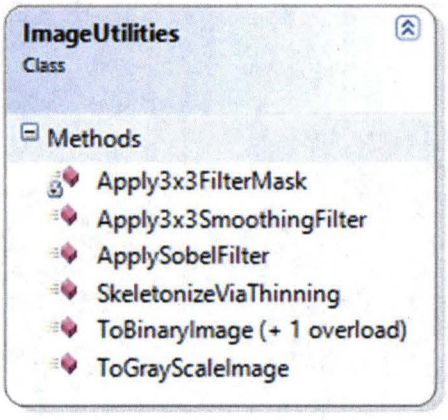



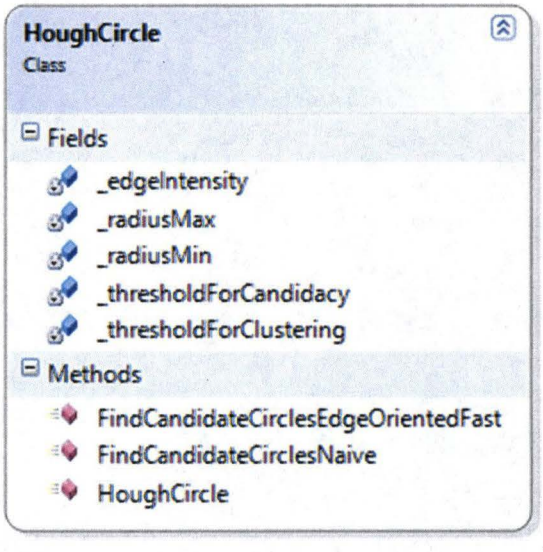


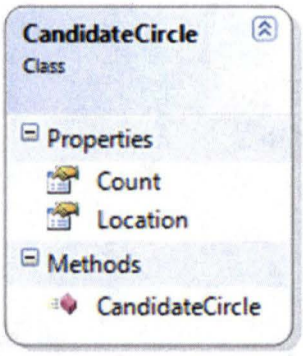



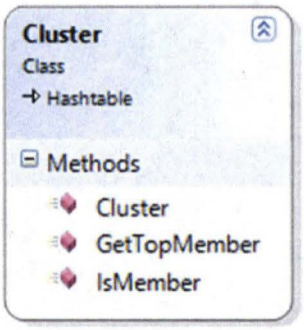



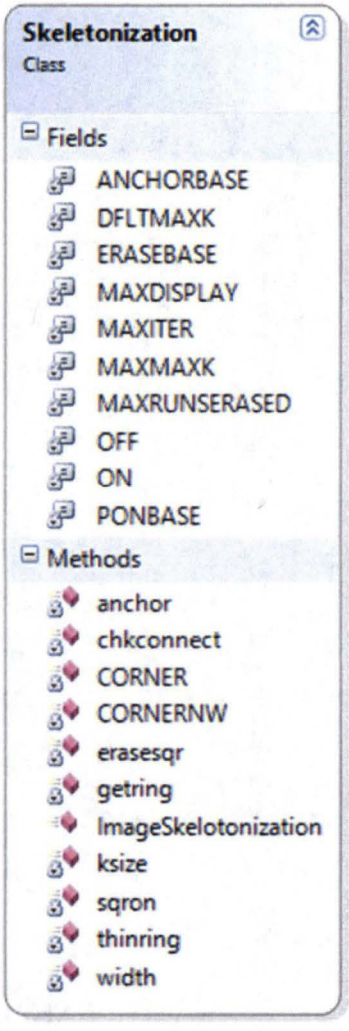

**D. Image Processing DLL Classes**

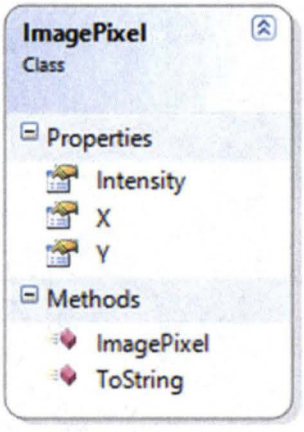


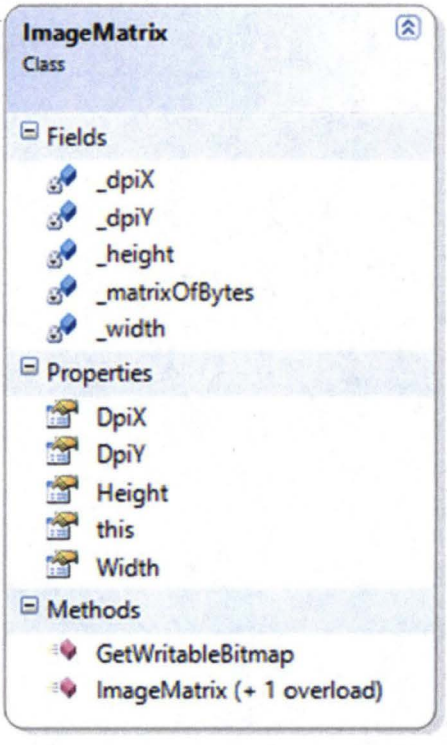



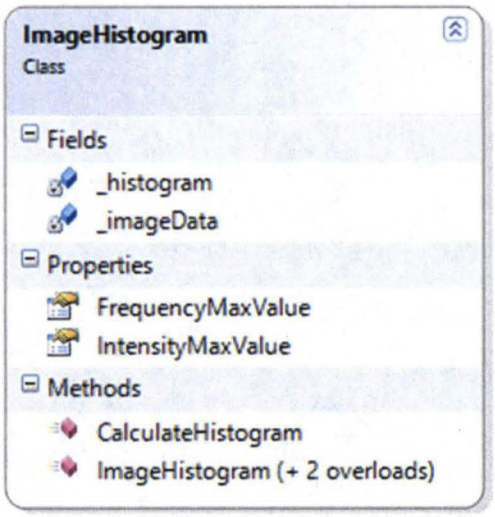

CandidateCircle
Class
Properties
Count
Location
Methods
CandidateCircle

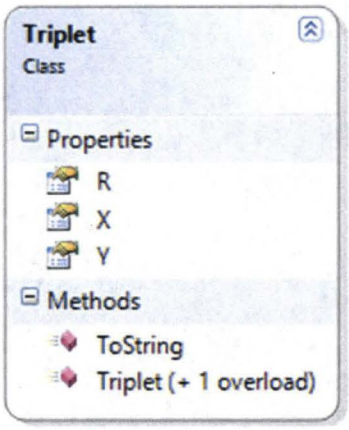

Cluster
Class
Hashtable
Methods
Cluster
GetTopMember
IsMember


Skeletonization
Class
Fields
ANCHORBASE
DFLTMAXK
ERASEBASE
MAXDISPLAY
MAXITER
MAXMAXK
MAXRUNSERASED
OFF
ON
PONBASE
Methods
anchor
chkconnect
CORNER
CORNERNW
erasesqr
getring
ImageSkelotonization
ksize
sqron
thinring
width


ImagePixel
Class
Properties
Intensity
X
Y
Methods
ImagePixel
ToString


ImageMatrix
Class
Fields
_dpiX
_dpiY
_height
_matrixOfBytes
_width
Properties
DpiX
DpiY
Height
this
Width
Methods
GetWritableBitmap
ImageMatrix (+ 1 overload)




## Acknowledgments

I would like to thank Barbara Brown, Felix Soto Toro, Charlie Goodrich and Bob Panzak for selecting me for this great opportunity. I would like to thank the NATIONAL SPACE GRANT FOUNDATION for providing the funding for my opportunity. I would like to thank Rose Austin, Benita Desuza, Grace Johnson, Rob Cannon and everyone in the KSC Education office for all their hard work in organizing and facilitating the internship experience. Finally, I would like to give an extra thank you to Felix for being a great mentor.