FAA Remote-Controlled Crack Monitoring (RCCM) Video System

Software Design

Mark Davis
Michael Bartholomew

Advisors
Dr. Werner Krandick, Computer Science
Dr. Jonathan Awerbuch, Mechanical Engineering
Dr. Tein-Min Lan, Mechanical Engineering
Dr. Alan Lau, Mechanical Engineering
Dr. John Bakuckas, FAA

March 2, 2007
Table of Contents

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Design Goals
   1.4 References
   1.5 Definitions

2. Design Overview
   2.1 Problem Description
   2.2 Technologies Used
   2.3 Application Overview

3. User Interface
   3.1 User Interface Overview
      3.1.1 Inputs and Operators
   3.2 Interface Specifications
      3.2.1 Interface Functional Map
      3.2.2 Camera/Motor Control
         3.2.2.1 Cartesian Control
         3.2.2.2 Velocity Control
      3.2.3 Camera View
         3.2.3.1 Measurement Mode
      3.2.4 Archiving View
      3.2.5 Settings
   3.3 User Interface Design
      3.3.1 Databinding
      3.3.2 Generic Input Devices Overview
      3.3.3 Generic Input Devices and Databinding Proof of Concept

4. Hardware-Software Interaction
   4.1 Component Overview
   4.2 Motor Controller
      4.2.1 Specifications
      4.2.2 Driver/SDK
   4.3 Video Capture Card
      4.3.1 Specifications
      4.3.2 Driver/SDK
   4.4 Optical Disc Recorder
      4.4.1 Specifications
      4.4.2 Driver/SDK
   4.5 Event Stamping Interface
      4.5.1 Specifications
      4.5.2 Driver/SDK

5. Software Architecture
1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed design description of the system that satisfies all of the sections in the requirements document. This document will contain specifications for developing the entire software system. Included are a description of the software and its desired purpose, user interface diagrams, and detailed, technical descriptions of the required components needed to accomplish the requirements.

1.2 Scope

This document contains a description of the design for the RCCM system software. It is a technical document intended for the developers of the system, as well as for a reference of the software design for future modifications. This document describes the necessary components of the RCCM system to perform the functional and non-functional requirements previously detailed in the requirements document.

The graphical user interface is shown as it will be seen by the user, along with diagrams describing the flow of the GUI. A high level description of the entire system, along with individual components, precedes any technical information. A UML diagram of the component and its individual objects, along with a reference to the requirements satisfied and a detailed description of each object and its attributes/operations is present to aid in the development of the software.

Due to the cyclical nature of software development, the initial design shown in this document could be changed during implementation. As a primary purpose of this document is to provide information for future developers modifying the system, the design document will be updated to reflect the final, implemented design.

1.3 Design Goals

The RCCM software described here is a new application replacing an existing system. The goal of the new software is to satisfy the following:

- **Ease of Use.** The user interface of the RCCM system is to be intuitive, efficient, and clean. It must support single or multiple operators.
- **Upgradability.** The FAA will be provided with all source code and design documents for this software system. The software must be easily modified to support new functionality and different hardware components.
- **All-digital operation.** The existing software does not contain any archiving functions; this is handled by an external analog system. The new software must provide the ability to archive all information digitally, both internally and externally on optical media.
• Accuracy. This software is used by the FAA in a project that is intended to save lives, so it is crucial that all information captured and calculated by the application is 100% accurate.

1.4 References

Microsoft .NET Framework

Oregon Microsystems MaxP driver software
<http://www.pro-dex.com/software_motion.php>

Matrox Imaging Library
<http://www.matrox.com/imaging/products/mil/home.cfm>

Nero COM SDK
<http://www.nero.com/nero7/enu/SDK_API.html>

FAA FASTER Facility

1.5 Definitions

Camera – Analog video cameras that monitor airplane fuselage panels, controlled with a video capture card in the PC

FAA – Federal Aviation Administration. Government agency responsible for regulating and developing aviation

FASTER – Full-Scale Aircraft Structural Test Evaluation and Research. The testing facility the RCCM video system monitors

GUI – Graphical User Interface

Motor – Translation stages that move the camera system. Controlled through a motor control card in the PC

RCCM – Remote-Controlled Crack Monitoring. The camera/motor system the software is intended to control

SDK – Software Development Kit. Software provided to allow a developer to perform specific functions, e.g. interface with hardware drivers
2. Design Overview

2.1 Description of Problem

The application must provide the user with accurate, real-time information and control over the RCCM system. It is important to maintain different design aspects during development, including performance, adherence to the requirements document, and to keep individual components decoupled from the core of the application. It is very likely the FAA will upgrade the hardware in the RCCM system in the future, and the software must be easily adaptable to this new hardware. The GUI must also be designed significantly better than in the original software. This will allow for more efficient operation, as well as accommodate for the additional features included in this application.

2.2 Technologies Used

The RCCM software is to be written using the .NET framework. Any of the languages supporting .NET are usable, although C# is preferred. The application also relies on several third party SDKs for interfacing with hardware. The Oregon Microsystems MaxP-8000 motor control card and Matrox Morphis video capture card use manufacturer-supplied SDKs. A number of third party libraries, such as the Nero SDK, can be used to output to an optical drive.

2.3 Application Overview

Once implemented, the application will be capable of fully controlling the RCCM camera system remotely. All translation stages in the system, currently eight, will be controlled and their status maintained by the software. The cameras in the system, currently five, will be viewable simultaneously, automatically recorded digitally, with the capability of taking individual still shots and video. Current test data, including pressure, cycle counts, and load, will be viewable by the software and stamped on all media. Additionally, the software will support either one or two concurrent users.
3. User Interface

3.1 User Interface Overview

The RCCM software interface must provide accurate information and manipulation of all relevant input and output sources. Since the system must interface with multiple hardware interfaces, and at the same time support multiple operators, the interface must be designed in a way where at any given time, the operator is able to get a real-time view of the state of the system. The context diagram below illustrates the complexities of dealing with such a system.

In response to this, a sophisticated architecture must be designed to cope with some of these complexities.
3.1.1 Inputs and Operators

The first challenge in such a system is to support a generic input system to allow for multiple operators on multiple input devices to operate concurrently, and secondly, to synchronize all data between the hardware components, the engine of the software, the interface layer, and the multiple clients which may control it – operators, or the input devices the operators are utilizing.

In the above diagram, we see that the interface has multiple clients it needs to support. Not only could there be multiple operators manipulating the system, the operators also may be channeling interface input through an input device. While handling standard devices like a keyboard or mouse is less complex, handling multiple keyboards or mice within the same system, or handling less standard devices like a joystick or pen based device can be complex.
3.2 Interface Specifications

This section describes each interface component and provides a high fidelity prototype of what the system will look like.

3.2.1 Interface Functional Map

A map of each interface function is provided below. Section 3.2.2 begins the outline of each interface form that is referenced in the below table.

<table>
<thead>
<tr>
<th>Interface Form</th>
<th>Component</th>
<th>Location</th>
<th>Opens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera/Motor Cartesian</td>
<td>Archive</td>
<td>Top Right</td>
<td>Archive</td>
</tr>
<tr>
<td>Camera/Motor Cartesian</td>
<td>Settings</td>
<td>Top Right</td>
<td>Settings</td>
</tr>
<tr>
<td>Camera/Motor Cartesian</td>
<td>Camera Switching</td>
<td>Top Middle</td>
<td>Selected Camera View</td>
</tr>
<tr>
<td>Camera/Motor Cartesian</td>
<td>Camera Config</td>
<td>Top Middle</td>
<td>Camera Configuration</td>
</tr>
<tr>
<td>Camera/Motor Cartesian</td>
<td>Input</td>
<td>Middle Left</td>
<td>Input Device Selection</td>
</tr>
<tr>
<td>Camera/Motor Cartesian</td>
<td>Velocity Switch</td>
<td>Middle Left</td>
<td>Velocity Control (IP)</td>
</tr>
<tr>
<td>Camera/Motor Velocity</td>
<td>Cartesian Switch</td>
<td>Middle Left</td>
<td>Cartesian Control (IP)</td>
</tr>
<tr>
<td>Camera View</td>
<td>Color Selection</td>
<td>Top Right</td>
<td>Color Dialog</td>
</tr>
<tr>
<td>Camera View</td>
<td>Playback</td>
<td>Bottom Right</td>
<td>Video Buffer &amp; Controls (IP)</td>
</tr>
<tr>
<td>Camera View</td>
<td>Live View</td>
<td>Bottom Right</td>
<td>Live View &amp; Status (IP)</td>
</tr>
<tr>
<td>Camera View</td>
<td>Crosshair</td>
<td>Top Right</td>
<td>Zero Based Meas (IP)</td>
</tr>
<tr>
<td>Camera View</td>
<td>Pen Add</td>
<td>Top Right</td>
<td>Point &amp; Click Meas (IP)</td>
</tr>
<tr>
<td>Camera View</td>
<td>Pen Draw</td>
<td>Top Right</td>
<td>Aided Measurement (IP)</td>
</tr>
<tr>
<td>Archive</td>
<td>Import</td>
<td>Bottom Right</td>
<td>Import File Dialog</td>
</tr>
<tr>
<td>Archive</td>
<td>Export</td>
<td>Bottom Right</td>
<td>Export File Dialog</td>
</tr>
<tr>
<td>Archive</td>
<td>Archive</td>
<td>Bottom Right</td>
<td>External Device Dialog</td>
</tr>
<tr>
<td>Settings</td>
<td>Input Device</td>
<td>Top Middle</td>
<td>Input Device Selection</td>
</tr>
<tr>
<td>Settings</td>
<td>Image Format</td>
<td>Middle Left</td>
<td>Image Format Selection</td>
</tr>
<tr>
<td>Settings</td>
<td>Image Size</td>
<td>Middle</td>
<td>Image Size Selection</td>
</tr>
<tr>
<td>Settings</td>
<td>Video Format</td>
<td>Bottom Left</td>
<td>Video Format Selection</td>
</tr>
<tr>
<td>Settings</td>
<td>Video Size</td>
<td>Bottom Middle</td>
<td>Video Size Selection</td>
</tr>
<tr>
<td>Settings</td>
<td>Frames Per Sec</td>
<td>Bottom Right</td>
<td>Video FPS Selection</td>
</tr>
</tbody>
</table>
3.2.2 Camera/Motor Control

The Camera/Motor Control provides the main interface for an operator to interact with the RCCM system.

3.2.2.1 Camera/Motor Cartesian Controls

The below image shows the control when standard, Cartesian translation of the selected motor system is selected (green arrows.)

- The **bottom left** section of the interface shows the translation controls for manipulating the selected motor/camera (top left section of the interface) along the X, Y, and Z axis.
- The **bottom tab** displays event data and time stamps received from the event hardware.
- The **bottom right** section of the interface illustrates the positional awareness control, showing the relative position of the various controls.
- **Above** the positional awareness control, we have buttons to bring up the settings and archive forms. In the top middle section of the interface, we have controls for displaying the various camera views, and configuration relevant to the cameras.
3.2.2.2 Camera/Motor Velocity Control

The below image shows the control when velocity translation of the selected motor system is selected (blue arrows.)

- The **bottom left** section of the interface shows the translation controls for manipulating the selected motor/camera’s (top left section of the interface) velocity along the X, Y, and Z axis.
- **Stop cues** for the translating motor are embedded within the relevant translation direction (denoted as the red stop arrow.)
3.2.3 Camera View

The Camera View control provides a view of the video buffer and video control, image/video capture, and various measurement taking utilities. Multiple camera views may be up at once on multiple monitors. Zero-based measurement is demonstrated in the below example.

- The **bottom left** section of the interface shows the currently selected camera view and RCCM system.
- To the **right** of this, we have controls for beginning and ending video and image captures.
- To the **right of the capture controls** the current position of the video buffer and size are displayed.
- The **bottom right** section of the interface shows playback and live view mode selection, as well as video playback controls for the video buffer.
- The **center** of the interface shows the actual video feed from the selected camera.
- The **top** section of the interface shows various measurement types (top left) and the options for the line/crosshair properties of measurements.
3.2.3.1 Camera View Measurement Mode

The top left section of the camera view interface demonstrates various measurement modes and graphical attributes. From left to right in top left corner, we see the zero-based measurement, point and click, and the crack tracing measurement modes.

- The center of the control shows a potential camera view, where a crack needs to be measured.
- In the top left section of the interface, the point and click measurement method is selected.
• In the **center** of the control, An operator has drawn lines along the crack, representing the measurement vector that will be computed.  
• Each **point** (as indicated in yellow) represents where the operator has clicked, with interpolation of lines between points. These lines will be translated into measurement segments as described in the following sections.
3.2.4 Archiving View

The Archive view control provides the interface for interacting with all of the archives contained within the system, and with any currently existing data (which is automatically archived.)

- The **bottom left** section of the interface shows the controls for adding, deleting, and clearing archive data.
- In the **bottom right** section, we have controls for importing, exporting, and archiving the selected data.
- The **center** of the interface lists the actual archive data, sortable by the various metadata tabs on the list control.
- In the **top left section** of the interface, flags for showing or hiding categories of archive data are shown.
- In the **top right** corner, we have a live search or filter for the data.
3.2.5 Settings

The RCCM system has various settings which can be configured by an operator, and will be persistently stored by the RcConfigurationManager (see below sections.)

• Simple interface for selecting the current input device.
• This input device will then map to the relevant controls for translation of the selected camera or motor. See databinding and input sections for more information.

• A subset of the RCCM system settings, showing various selections for input device, image, and video settings.
3.3 User Interface Design

Several mechanisms were designed to deal with the issues of synchronization, data concurrency, and generic input systems. The below section outlines the overall design and patterns used.

3.3.1 Databinding

To maintain concurrency between all of the data within the system, a RcBindingManager (see engine architecture section) was designed to aid in advanced databinding, using primarily PropertyManager style databinding (seen in the diagram below on the right.)

The input arrows shown above between the gray boxes represent flow of data when two properties are databound. In the example above, the property on MyObject is bound to the DataBound property through the Binding Object. The Binding Object then maintains concurrent updating of properties between each object. If MyObject’s property changes, the Data Bound property is automatically updated. Similarly, if the Data Bound property changes, so will the property on MyObject. The RcBindingManager provides options to allow for both two way and one way databinding, meaning that either both, or just one of the synchronization relations just mentioned will hold, respectively.
3.3.2 Generic Input Devices Overview

Since input needs to be made generic through the system, the \texttt{RcInput} schema was designed to manage generic inputs and appropriate mappings. Instead of a control mapping to standard .NET keyboard and mouse events, they will subscribe to events on an RcInput instance that is contained on the base class of every user interface control in the RCCM system (\texttt{RcUiControl}). The base class, RcInput is listed below,
The System allows for generic input events, as listed above, which can then be implemented in terms of any input device, which is then implemented in terms of RcInput. The standard RcKeyboardInput and RcMouseInput, listed below, both inherit from RcInput. They then subscribe to .NET keyboard and mouse events, and forward the events to the generic events on RcInput.

The RcInputMapping, as shown below, provides special mapping for special case inputs like the keyboard or joystick, where the device outputs may not be in a standard form (such as the one RcInput implements.) For instance, the class can provide mappings between the arrow keys on the keyboard, or the up/down/left/right motions of a joystick and the generic event providers on RcInput. User customizable mappings will allow for mapping different keys or joystick motions to different input events.
The **RcInputDecorator** (which also inherits from RcInput) allows for multiple Input context to be mapped to the same RcInput. For instance, a RcUiControl may wish to instantiate its RcInput with an RcInputDecorator consisting of both an RcInputMouse and an RcInputKeyboard. The RcUiControl then subscribes to the generic events on the RcInput. Subsequently, the two different devices will work in their own way to manipulate the control, which is only implemented in terms of the generic input schema in RcInput.

**RcInputChain**, a type of RcInputDecorator, forwards input events between RcInputs. This allows for a RcUiControl which contains child RcUiControls to establish a RcInputChain that will forward all input events from the child controls to the parent.
3.3.3 A Generic Inputs and Databinding Proof of Concept

With the RcUiCartesianTranslation control, which is shown below and is seen in the Camera/Motor control interface specifications, we map it’s aggregate RcInput (from RcUiControl) to received keyboard events, and provide the keyboard event mapper with appropriate key translations using the RcInputMapping.

When then subscribe to the generic RcInput events in the RcUiCartesianTranslation: in this case, InputXDecrement, InputXIncrement, InputYDecrement, and InputYIncrement. The class diagram is shown below.
The event handlers then appropriately translate the Coordinate aggregate within RcUiCartesianTranslation, providing Cartesian translation of the coordinate.

In the engine, we use the **RcDatabindingManager** to bind the RcCoordinate properties of both RcUiCartesianTranslation and the selected RcMotor. We also bind the RcMotor to the Coordinate that is aggregate within another class, **RcUiPositionDisplay**, as seen below and in the motor/camera control interface specifications.

When an operator provides an input event, the **RcUiCartesianTranslation** then adjusts it’s coordinate, which then (because of the databinding,) adjust the coordinate on the **RcMotor**, which translates the motor, which then (because of databinding,) adjusts the coordinate on the **RcUiPositionDisplay**, which appropriately updates its display.
4. Hardware-Software Interaction

4.1 Component Overview

The purpose of this software is to control the hardware in the RCCM camera system. To accomplish this, a large portion of the project revolves around interfacing the software with the RCCM components, through a variety of control cards. There are four hardware interfaces:

Motor – the eight translation stages must be controlled via the software. Required functionality includes moving each motor individually, monitoring the current location of the RCCM system, and preventing over-traveling. To do this, the software must track the location of each stage at all times, as well as the limits of motion in all directions.

Video – the five analog cameras must be accessible to the user. This includes viewing all cameras simultaneously, recording the video feed, and taking digital still shots. Also, the software must be fully configurable to store these video and images in a variety of different formats.

Optical Drive – at all times, the software maintains an archive of video, image, and other data objects recorded by the application. This archived data must be periodically stored on optical discs, for redundancy, as well as due to space concerns on the PC. These optical discs could be either CD or DVD, and various media file system formats must be supported.

Event Stamping – as the FASTER tests progress, it is necessary for the software to display the current progress of the test. Also, this information must be visually stamped on all recorded media, and be stored in the archive.

The software will be initially implemented for specific hardware; however, it is crucial to maintain a decoupled design between the hardware and the application, so these hardware devices can be easily changed.

4.2 Motor Controller

4.2.1 Specifications

The motor control card initially used is the Oregon Microsystems MAXp-8000 Multi-Axis Motion Controller. This card is capable of simultaneous control of up to eight axes. It is a very powerful card, with a dedicated 266MHz PowerPC RISC processor built in. The motors currently used by the RCCM system are simple stepper motors; however, this controller is capable of controlling any combination of stepper and servo motors.

4.2.2 Driver/SDK
Oregon Microsystems provides drivers and a full-featured SDK for the MAXp-8000. One small caveat is that this SDK is written for a C++ development environment. To make this SDK compatible with the RCCM software, a wrapper must be created. This is a very simple, one-one function wrapper that maps calls to a C# dynamic library to the supplied C++ SDK. An example communication sequence is shown below:

4.3 Video Capture Card

4.3.1 Specifications

The analog video capture card used will be the Matrox Morphis QxT 16. This card is capable of capturing up to 16 simultaneous video feeds. It has 256MB of on-board memory, including 128MB of frame buffer memory and 128MB of processing memory. The board can compress incoming video on-the-fly with its on-board processing unit.

4.3.2 Driver/SDK

The software will use the MIL8, or Matrox Imaging Library 8, to manipulate the video captured by the Morphis card. The MIL is fully compatible with the .NET architecture the RCCM software uses. The SDK also fully uses all available hardware acceleration for maximum performance, including the Morphis’ onboard processing, and Intel’s MMX/SSE/SSE2 extensions. The MIL has functions for complete control over the Morphis card, including image/video capturing, processing/converting video, analysis, and display.

4.4 Optical Disc Recorder

4.4.1 Specifications

Archiving data to an optical disc is a much more generic process than the other hardware-software interactions in the RCCM system. Any recordable DVD/CD drive can be used to accomplish this task. There are a variety of third party tools that can then be used to interface the RCCM software with this drive. The default for the application will be the Nero SDK. Nero works in conjunction with the Nero recording software, and provides a very powerful SDK for application development.
4.4.2 Driver/SDK

The Nero SDK operates independently of the hardware it controls. Any DVD/CD can be used in conjunction with the SDK. One object in the Nero software can be used to burn either a CD or a DVD, with a few simple attribute changes. In addition, Nero provides a sweet of example applications that make adapting the Nero SDK to the RCCM software a very easy, well documented task.

4.5 Event Stamping Interface

4.5.1 Specifications

Event stamping is one of the most crucial parts of the RCCM system. The external device that controls the test interfaces with the RCCM software through a National Instruments NI 6025E Multifunction Analog Data Acquisition card. This card is capable of receiving up to 16 analog inputs. Through this card, the RCCM software can continuously monitor the test status, including current pressure, cycle count, and load.

4.5.2 Driver/SDK

The NI 6025E communications to the RCCM software indirectly through a set of driver applications. In the driver application, the user sets which inputs are monitoring and their settings. A simple API known as NI-DAQ allows the RCCM software to monitor the status of this driver, which can then can update the user with this information in real time, as well as stamp it on all recorded media.
5. System Architecture
Engine

The RCCM engine consists of various objects and communication schemas that comprise the functionality which allows for class interaction, databinding between objects, and generic input systems throughout the system. The system also provides conduits for archiving, data synchronization, and static access to system components. Archiving, databinding, RCCM system/component interactions, and attribute persistence is all achieved through the general engine architecture.

RcEngine Class

Provides a static placeholder to aid in class communication, databinding, and global access to archiving functionality.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>RcSystem</td>
<td>Static access to RcSystem object, providing communication with Rccm cameras, motor systems, event providers, and configuration options.</td>
</tr>
<tr>
<td>Archive</td>
<td>RcArchiveManager</td>
<td>Static access to RcArchiveManager, providing communication with archiving capability. Acts as conduit to provide system wide archiving of media and measurement objects as they are instantiated.</td>
</tr>
<tr>
<td>Binding</td>
<td>RcBindingManager</td>
<td>Static access to RcBindingManager, providing databinding and synchronization between classes and data members within the Rccm system.</td>
</tr>
<tr>
<td>Configuration</td>
<td>RcConfigurationManager</td>
<td>Static access to RcConfigurationManager, providing access to attribute/configuration persistence between application runs.</td>
</tr>
</tbody>
</table>
RcBindingManager Class

Provides helpers to aid in supplementing databinding and general data synchronization between system components and properties. The RcBindingManager is the foundation of synchronization architecture; see section 3.3 for more information.

RcBindingManager

+Bind(Source: object, Destination: object, BindingContext: eBindingContext)
+Clear(Source: object, Destination: object): void

Operations

void Bind (object Source, string SourceProperty, object Destination, string DestinationProperty)
Inputs: Source – class source to bind from
         SourceProperty – class source property to bind from
         Destination – class destination to bind
         DestinationProperty – class destination property to bind source to
Outputs: None
Description: Binds indicated properties on provided classes to provide complete data synchronization between the two.

void Bind (object Source, object Destination)
Inputs: Source – class source to bind from
         Destination – class destination to bind to
Outputs: None
Description: Uses reflection to bind two instances of the same class to provide complete data synchronization between the two.

void Clear (object Source, string SourceProperty, object Destination, string DestinationProperty)
Inputs: Source – class from which to clear databinding link from
         SourceProperty – the property which is to be unbound.
         Destination – class from which the databinding is linked
         DestinationProperty – the property on the destination which is to be unbound.
Outputs: None
Description: Clears databinding synchronization between indicated classes.

void Clear (object Source, object Destination)
Inputs: Source – class from which to clear databinding link from
Destination – class from which the databinding is linked

Outputs: None
Description: Clears databinding synchronization between indicated classes.

---

**RcConfigurationManager Class**

Provides saving and loading of system wide configuration options. Used for maintaining options, settings, and other information between application runs.

```
<table>
<thead>
<tr>
<th>RcConfigurationManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path: string</td>
</tr>
<tr>
<td>Load(): void</td>
</tr>
<tr>
<td>Save(): void</td>
</tr>
</tbody>
</table>
```

**Requirements Satisfied**

5.1.5 5.2.3

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>string</td>
<td>Internal use property indicating the path to the configuration file.</td>
</tr>
</tbody>
</table>

**Operations**

**void Save ()**

Inputs: None
Outputs: None
Description: Uses static communication foundation in engine to serialize all relevant options and attributes that need to stay persistent between application runs.

**void Load ()**

Inputs: None
Outputs: None
Description: Uses static communication foundation in engine to deserialize and set all relevant options and attributes that were saved with Save().
RcContext Class

Interacts with the event provider hardware to provide information on the current system time, cycles, pressure, applied load, and other RCCM system test information. *RcNI6025EContext provides the implementation for the National Instruments data acquisition hardware (see section 4.5.1)*

![RcContext class](image)

Requirements Satisfied

4.2.3  4.3.1  4.3.2  4.3.4  4.3.6  4.3.8

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>DateTime</td>
<td>The current system time.</td>
</tr>
<tr>
<td>Cycles</td>
<td>double</td>
<td>The RCCM test system cycle count.</td>
</tr>
<tr>
<td>Pressure</td>
<td>double</td>
<td>The current pressure exerted on the RCCM test system.</td>
</tr>
<tr>
<td>Load</td>
<td>double</td>
<td>The applied load on the RCCM test system.</td>
</tr>
</tbody>
</table>

RcSystem Class

Placeholder for all components related to the RCCM System. This includes the dual RCCM camera/motor systems, the structure motor system, and the current system context (derived from the event provider hardware interface), and the static, ultra wide field of view (UWFOV) surveillance camera.
Requirements Satisfied

4.1.2  4.1.7  4.1.8  4.1.13

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RccmStructure</td>
<td>RcMotorSystem</td>
<td>Access to the motor system controlling the overlying RCCM structure.</td>
</tr>
<tr>
<td>Rccm1</td>
<td>RcMotorSystem</td>
<td>Access to the first RCCM motor/camera system.</td>
</tr>
<tr>
<td>Rccm2</td>
<td>RcMotorSystem</td>
<td>Access to the second RCCM motor/camera system.</td>
</tr>
<tr>
<td>UltraWideFOV</td>
<td>RcCamera</td>
<td>Access to the static, ultra wide field of view (UWFOV) surveillance camera.</td>
</tr>
<tr>
<td>Context</td>
<td>RcContext</td>
<td>Access to RcContext, the event provider interface which communicates with event provider hardware to provide information about the current state of the system.</td>
</tr>
</tbody>
</table>

RcLimit Class

Provides constraints or bounds for motors and other relevant controls utilizing a coordinate system.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>RcCoordinate</td>
<td></td>
</tr>
<tr>
<td>Top</td>
<td>RcCoordinate</td>
<td></td>
</tr>
<tr>
<td>Right</td>
<td>RcCoordinate</td>
<td></td>
</tr>
<tr>
<td>Bottom</td>
<td>RcCoordinate</td>
<td></td>
</tr>
</tbody>
</table>

Requirements Satisfied

4.1.3  5.3.2

Attributes
Left | RcCoordinate | The left-most bound coordinate.
Top  | RcCoordinate | The top-most bounds coordinate.
Right| RcCoordinate | The right-most bounds coordinate.
Bottom| RcCoordinate | The bottom-most bounds coordinate.

### RcCoordinate Class

Represents a location within a coordinate system in the RCCM application.

<table>
<thead>
<tr>
<th>RcCoordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Planar: bool</td>
</tr>
<tr>
<td>+X: double</td>
</tr>
<tr>
<td>+Y: double</td>
</tr>
<tr>
<td>+Z: double</td>
</tr>
</tbody>
</table>

### Requirements Satisfied

4.1.3 4.1.5

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>double</td>
<td>The location along the X axis.</td>
</tr>
<tr>
<td>Y</td>
<td>double</td>
<td>The location along the Y axis.</td>
</tr>
<tr>
<td>Z</td>
<td>double</td>
<td>The location along the Z axis.</td>
</tr>
<tr>
<td>Planar</td>
<td>bool</td>
<td>Indicates whether the coordinate has a Z component or is planar.</td>
</tr>
</tbody>
</table>

### RcMotor Class

Provides interface for manipulating translation stages through motor, and mapping motor revolutions to a standard coordinate system. The motor class handles overall motor bounds or limits, provides general position information and motor translation, and allows for automating forward velocity in any of the axial directions. Also provides relevant calibration, resetting, and initialization of motor. 

*RcMaxP8000Motor provides the implementation for the Oregon Microsystems Motor hardware (see section 4.2.1.)*
Requirements Satisfied

4.1.1  4.1.2  4.1.3  4.1.4  4.1.5  4.1.6  4.1.7  4.1.10  4.1.11  4.1.12  4.1.13  4.1.14  4.1.15

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>RcCoordinate</td>
<td>The current position of the motor in the relevant coordinate system.</td>
</tr>
<tr>
<td>Velocity</td>
<td>RcCoordinate</td>
<td>The current forward axial velocities. All translation within the system will apply relevant velocities to translation.</td>
</tr>
<tr>
<td>Limit</td>
<td>RcLimit</td>
<td>Provides the translation bounds of the motor.</td>
</tr>
</tbody>
</table>

Operations

**void Motor (bool bPlanar)**

Inputs: bPlanar – Boolean indicating whether the camera exists on a planar axis or not.

Outputs: None

Description: Constructs and initializes a motor to the indicated coordinate System.

**void Move (double X, double Y, int Velocity)**

Inputs: X – The X translation step.
Y – The Y translation step.
Velocity – The forward velocity of the translation.

Outputs: None
Description: Translates the motor with the indicated inputs.

void Move (double X, double Y, double Z, int Velocity)
Inputs: X – The X translation step.
        Y – The Y translation step.
        Z – The Z translation step.
        Velocity – The forward velocity of the translation.
Outputs: None
Description: Translates the motor with the indicated inputs.

void MoveX (int Velocity)
Inputs: Velocity – The forward velocity of the translation.
Outputs: None
Description: Helper function that translates the motor in the X axial direction with a forward velocity.

void MoveY (int Velocity)
Inputs: Velocity – The forward velocity of the translation.
Outputs: None
Description: Helper function that translates the motor in the Y axial direction with a forward velocity.

void MoveZ (int Velocity)
Inputs: Velocity – The forward velocity of the translation.
Outputs: None
Description: Helper function that translates the motor in the Z axial direction with a forward velocity.

void Stop ()
Inputs: None
Outputs: None
Description: Terminates all current translation in all axial directions.

void StopX ()
Inputs: None
Outputs: None
Description: Terminates all current translation in all X axial direction.

void StopY ()
Inputs: None
Outputs: None
Description: Terminates all current translation in all Y axial direction.

void StopZ ()
void Reset ()
 Inputs: None
 Outputs: None
 Description: Resets the motor to its default state.

void Calibrate ()
 Inputs: None
 Outputs: None
 Description: Calibrates the motor utilizing the current location and coordinate system.

**RcMotorSystem Class**

Base class for all motor systems, which include all RCCM camera systems. Provides communication with underlying RcMotor.

**Requirements Satisfied**

4.1.1  4.1.2  4.1.3  4.1.4  4.1.5  4.1.6  4.1.7  4.1.10  4.1.11  4.1.12  4.1.13  4.1.14  4.1.15

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor</td>
<td>RcMotor</td>
<td>Access to the motor controlling translation for the given motor system.</td>
</tr>
</tbody>
</table>

**RcCameraSystem Class**
Derives from RcMotorSystem for motor communications. Provides access to both cameras constituting a RCCM camera/motor system: the narrow field of view (NFOV) camera, and the wide field of view (WFOV) camera.

**RcCamera System**

```
RcCameraSystem
+NarrowFOV: RcCamera
+WideFOV: RcCamera
```

Requirements Satisfied

4.1.8  4.1.9  4.1.10

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NarrowFOV</td>
<td>RcCamera</td>
<td>Access to the narrow field of view (NFOV) camera mounted on each RCCM system.</td>
</tr>
<tr>
<td>WideFOV</td>
<td>RcCamera</td>
<td>Access to the wide field of view (WFOV) camera mounted on each RCCM system</td>
</tr>
</tbody>
</table>

**RcCamera Class**

Provides main operations and attributes for camera communications. The RcCamera class provides the relevant Image and Video feeds from the camera, as well as operations for initializing, resetting, calibrating, and shutting down the camera.

```
RcCamera
+Image: RcImage
+Video: RcVideo
+Initialise(): void
+Shutdown(): void
+Reset(): void
+Calibrate(): void
```

Requirements Satisfied

4.2.2  4.2.7  4.2.11

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>RcImage</td>
<td>Access to the image feed from the camera.</td>
</tr>
<tr>
<td>Video</td>
<td>RcVideo</td>
<td>Access to the video feed from the camera.</td>
</tr>
</tbody>
</table>
void Initialize ()
    Inputs:  None
    Outputs: None
    Description: Initializes the camera.

global void Shutdown ()
    Inputs:  None
    Outputs: None
    Description: Shuts down the camera.

global void Reset ()
    Inputs:  None
    Outputs: None
    Description: Resets the camera to its default state.

global void Calibrate ()
    Inputs:  None
    Outputs: None
    Description: Calibrates the camera utilizing the current location and coordinate system.

RcImage Class

Provides main operations and attributes for image feeds within the RCCM system. Image format, size, depth and archiving/display functionality all comprise the class.
RcMorphis16Image provides the image implementation for the Matrox camera system (see section 4.3.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

Requirements Satisfied

4.2.10 4.2.11 4.2.12

Attributes
**ImageFormat** | **RcImageFormat** | Allows configuration and conversion of the image format.
---|---|---
**ImageSize** | eImageSize | Enumeration specifying the image size of the underlying image.
**ImageDepth** | eImageDepth | Enumeration specifying the image depth of the underlying image.

## Operations

**RcImage Capture ()**
- **Inputs:** None
- **Outputs:** RcImage – The captured Image, instantiated the indicated format, size, and depth.
- **Description:** Captures an image from the Camera and returns the image utilizing the image settings in the class.

**void Display ()**
- **Inputs:** None
- **Outputs:** None
- **Description:** Displays an image in its own window, utilizing the image settings present in the class.

**RcVideo Class**

Provides main operations and attributes for video feeds within the RCCM system. Video format, size, depth, and frames per second, as well as archiving/capture functionality all comprise the class. The RcVideo class also contains additional information about the video feed, such as it’s length, frame position, frame count, and buffer constraints. 

*RcMorphis16Video provides the video implementation for the Matrox camera system (see section 4.3.)*

```c
RcVideo
+VideoFPS: int
+VideoFormat: RcVideoFormat
+VideoSize: eVideoSize
+VideoDepth: eVideoDepth
+Length: DateTime
+Position: DateTime
+FrameCount: long
+BufferSize: long
+Capture(): void
+EndCapture(): RcVideo
```

## Requirements Satisfied

4.2.2 4.2.7 4.2.11
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VideoFPS</td>
<td>int</td>
<td>Allows configuration of the number of frames per second the video feed will contain.</td>
</tr>
<tr>
<td>VideoFormat</td>
<td>RcVideoFormat</td>
<td>Allows configuration and conversion of the video format.</td>
</tr>
<tr>
<td>VideoSize</td>
<td>eVideoSize</td>
<td>Enumeration specifying the video size of the underlying image.</td>
</tr>
<tr>
<td>VideoDepth</td>
<td>eVideoDepth</td>
<td>Enumeration specifying the video depth of the underlying image.</td>
</tr>
<tr>
<td>Length</td>
<td>DateTime</td>
<td>The total length of the video feed.</td>
</tr>
<tr>
<td>Position</td>
<td>DateTime</td>
<td>The current position of the video feed.</td>
</tr>
<tr>
<td>FrameCount</td>
<td>long</td>
<td>The total frame count of the video feed.</td>
</tr>
<tr>
<td>BufferLimit</td>
<td>long</td>
<td>Indicates the number of frames the buffer will store at any given time.</td>
</tr>
</tbody>
</table>

Operations

```plaintext
void Capture ()
Inputs: None
Outputs: None
Description: Begins capturing a video feed utilizing the RcVideo class settings.
```

```plaintext
RcVideo EndCapture ()
Inputs: None
Outputs: RcVideo – The captured video, utilizing the RcVideo class settings.
Description: Ends the capture of the video begun when Capture() was called. Returns the relevant RcVideo that was obtained during the capture, formatted using the various class settings in RcVideo.
```

RcVideoBuffer Class

Provides buffering and playback of an RcVideo within the RCCM application. Contains standard video playback operations, as well as operations for displaying the video.
Requirements Satisfied

4.2.2  4.2.7  4.2.11

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video</td>
<td>RcVideo</td>
<td>The underlying RcVideo which the buffer will work with.</td>
</tr>
</tbody>
</table>

Operations

**void RcVideoBuffer (RcVideo Video)**

Inputs: Video – The RcVideo which the buffer will utilize.
Outputs: None
Description: Constructs the RcVideoBuffer with the supplied RcVideo feed.

**void Play ()**

Inputs: None
Outputs: None
Description: Begins playback of the underlying RcVideo.

**void Pause ()**

Inputs: None
Outputs: None
Description: Pauses playback of the underlying RcVideo.

**void Stop ()**

Inputs: None
Outputs: None
Description: Stops playback of the underlying RcVideo.

**void Rewind ()**

Inputs: None
Outputs: None
Rewinds underlying RcVideo to its first frame.

```cpp
void Forward()
Inputs: None
Outputs: None
Description: Progresses underlying RcVideo to its last frame.
```

Steps the underlying RcVideo forward a frame.

```cpp
void StepForward()
Inputs: None
Outputs: None
Description: Steps the underlying RcVideo backward a frame.
```

Displays the RcVideo in its own window using the RcVideo settings, and provides interface for manipulation of buffer operations.

```cpp
void Display()
Inputs: None
Outputs: None
Description: Displays the RcVideo in its own window using the RcVideo settings, and provides interface for manipulation of buffer operations.
```

**Archive Manager**

The Archive Manager is responsible for all archive functions of the RCCM system. The system displays all archived information to the user, can be sorted/filters/searched, files in the archive can be accessed either internally or externally, and the contents of the archive can be exported to a CD/DVD or to a file. All measurements, images, and videos are automatically moved to the archive upon instantiation.

**RcArchiveManager Class**

The RcArchiveManager Class is the base of the archive system. It is responsible for providing a datasource for the contents of the archive, accessing and manipulating the archive items it contains, importing and exporting of data, and communication with the file and optical archivers.
Requirements Satisfied

4.3.1  4.3.2  4.3.3  4.3.4  4.3.5  4.3.6  4.3.7  4.3.8  4.3.9  4.3.10

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archives</td>
<td>ArrayList&lt;RcArchives&gt;</td>
<td>An arraylist of the archives currently loaded into the ArchiveManager</td>
</tr>
<tr>
<td>Archiver</td>
<td>RcArchiver</td>
<td>Utilized for archiving files to an external source, such as a filesystem or optical device.</td>
</tr>
</tbody>
</table>

Operations

RcArchiveManager ()

Inputs: None  
Outputs: None  
Description: Creates new ArchiveManager, loads archive from default file

RcArchiveManager (string Filename)

Inputs: Filename – file to load archive from  
Outputs: None  
Description: Creates a new ArchiveManager, loads archive from input file

RcArchive Add (RcImage Image)

Inputs: Image – RcImage to be archived.  
Outputs: Added archive.  
Description: Constructs a new RcImageArchive and adds it to the Archive list.

RcArchive Add (RcVideo Video)

Inputs: Video – RcVideo to be archived.  
Outputs: Added archive.  
Description: Constructs a new RcVideoArchive and adds it to the Archive list.
RcArchive Add (RcMeasurement Measurement)
Inputs: Measurement – RcMeasurement to be archived.
Outputs: Added archive.
Description: Constructs a new RcMeasurementArchive and adds it to the Archive list.

RcArchive Remove(int nIndex)
Inputs: nIndex – index in array of item to be removed
Outputs: Removed archive
Description: Removes the requested archive from the array; optionally removes from hard disk completely. Removed item is returned.

ArrayList Clear()
Inputs: None
Outputs: Current archive list
Description: Removes all items from archive; optionally removes from hard disk completely. List is returned.

RcArchive Import (string sFilename)
Inputs: sFilename – path of archive
Outputs: The archive if successfully loaded; otherwise null
Description: Loads archive information from the requested file.

RcArchive Export (int Index, string sFilename)
Inputs: Index – the archive to be exported
Outputs: returns exported archive if successful; otherwise null
Description: Exports an existing archive to a specific location.

RcArchive Operator[] (string sString)
Inputs: sString – String identifying archive
Outputs: The archive if successfully retrieved; otherwise null
Description: Provides indexing into internal Archive list, returns Archive.

RcArchive Operator[] (int nIndex)
Inputs: nIndex – Index identifying archive
Outputs: The archive if successfully retrieved; otherwise null
Description: Provides indexing into internal Archive list, returns Archive.

RcArchiver Class

Base class for backing up archive data to external destinations. The class optionally supports generic compression of archives. Each derived class must implement functionality to allowing for writing an RcArchive to its destination.
Requirements Satisfied

4.3.11

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression</td>
<td>Bool</td>
<td>If the files are to be compressed before archiving</td>
</tr>
</tbody>
</table>

Operations

**void Archive(RcArchive Archive)**
- Inputs: Archive – The RcArchive to be archived
- Outputs: None
- Description: Archives a specific archive to an external destination. Must be implemented by derived classes.

**void Archive(ArrayList<RcArchive> Archives)**
- Inputs: Archives – The RcArchives to be archived
- Outputs: None
- Description: Archives a list of archives to an external destination

**string Compress(RcArchive Archive)**
- Inputs: Archive - The RcArchive to compress
- Outputs: The serialized, compressed archive.
- Description: Compresses individual files in Archives

**RcOpticalArchiver Class**
RcOpticalArchiver extends the RcArchive class. It is an implementation for burning to a DVD or CD using a 3rd party library such as the Nero SDK. RcNeroOpticalArchiver provides the archiving implementation for the Nero SDK (see section 4.4.)

Requirements Satisfied

4.3.11

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrivePath</td>
<td>String</td>
<td>Drive to output with</td>
</tr>
<tr>
<td>DiscTitle</td>
<td>String</td>
<td>The title of the disc</td>
</tr>
<tr>
<td>DiscType</td>
<td>eDiscType</td>
<td>Type of media used: CD/DVD</td>
</tr>
<tr>
<td>DiscFormat</td>
<td>eDiscFormat</td>
<td>File format to be used on media</td>
</tr>
<tr>
<td>BurnSpeed</td>
<td>eBurnSpeed</td>
<td>Speed at which to burn media</td>
</tr>
</tbody>
</table>

Operations

**RcOpticalArchiver ()**
- Inputs: None
- Outputs: None
- Description: Initializes RcOpticalArchiver to use a standard DVD

**RcOpticalArchiver (string sTitle, string sDrive, eDiscType DiscType, eDiscFormat DiscFormat, eBurnSpeed BurnSpeed)**
- Inputs: sTitle – Title of the CD/DVD
  sDrive – Drive to output to
  DiscType – CD or DVD
  DiscFormat – File format of disc to be burned
  BurnSpeed – Speed to burn disc
- Outputs: None
- Description: Initializes RcOpticalArchiver with indicated values

**void Archive (RcArchive Archive)**
- Inputs: Archive – Archive to be written
- Outputs: None
Description: Outputs file to specified media

`void Verify ()`
Inputs: None
Outputs: None
Description: Checks if device settings are compatible with the output drive, and If disc is compatible with burn settings. Throws.

**RcFileArchiver Class**

Extends the RcArchiver class. It outputs an archive to a specified location within a filesystem, whether on a local or external device.

Requirements Satisfied

4.3.11

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>String</td>
<td>Absolute path to location where archives will be stored.</td>
</tr>
</tbody>
</table>

Operations

**RcFileArchiver ()**
Inputs: None
Outputs: None
Description: Initializes RcFileArchiver object to a “default” file.

**RcFileArchiver (string sPath)**
Inputs: sPath – absolute path to output archives to
Outputs: None
Description: Initializes RcFileArchiver path

**void Archive (RcArchive Archive)**
Inputs: Archive – Archive to be written
Outputs: None
Description: Outputs files to specified location
RcArchive Class

RcArchive defines the interface for storing information about a single item in the archive. It provides serialization and de-serialization, as well as a context from the event provider hardware with information about when the archive was instantiated.

Requirements Satisfied

4.3.3  4.3.5  4.3.6  4.3.7  4.3.8

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>RcContext</td>
<td>Test data at time of archive</td>
</tr>
<tr>
<td>Size</td>
<td>Double</td>
<td>Size of archive</td>
</tr>
</tbody>
</table>

Operations

**string Serialize()**

Inputs: None
Outputs: None
Description: Abstract. Serializes an archive for storage.

**void Deserialize(sSerial)**

Inputs: sSerial – The string of serialized archive.
Outputs: None
Description: Abstract. De-serializes an archive from storage and populates the archive class.

RcImageArchive Class
RcImageArchive extends the RcArchive class. It contains an RcImage object.

Requirements Satisfied

4.3.1  4.3.2  4.3.3  4.3.5  4.3.7

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>RcImage</td>
<td>The reference or copy of the aggregate RcImage object contained within the archive.</td>
</tr>
</tbody>
</table>

Operations

**string Serialize()**
- Inputs: None
- Outputs: None
- Description: Serializes an image archive for storage.

**void Deserialize(sSerial)**
- Inputs: sSerial – The string of serialized archive.
- Outputs: None
- Description: De-serializes an image archive from storage and populates the archive class.

RcVideoArchive Class

RcVideoArchive extends the RcArchive class. It contains an RcVideo object.

Requirements Satisfied

4.1.1  4.3.2  4.3.3  4.3.5  4.3.7

Attributes
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video</td>
<td>RcVideo</td>
<td>The reference or copy of the aggregate RcVideo object contained within the archive.</td>
</tr>
</tbody>
</table>

Operations

**string Serialize()**
- **Inputs:** None
- **Outputs:** None
- **Description:** Serializes a video archive for storage.

**void Deserialize(sSerial)**
- **Inputs:** sSerial – The string of serialized archive.
- **Outputs:** None
- **Description:** De-serializes a video archive from storage and populates the archive class.

**RcMeasurementArchive Class**

RcMeasurementArchive extends the RcArchive class. It contains an RcMeasurement object.

Requirements Satisfied

4.3.1  4.3.2  4.3.3  4.3.6

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement</td>
<td>RcMeasurement</td>
<td>The reference or copy of the aggregate RcMeasurement object contained within the archive.</td>
</tr>
</tbody>
</table>

Operations

**string Serialize()**
- **Inputs:** None
- **Outputs:** None
Description: Serializes a measurement archive for storage.

```java
void Deserialize(sSerial)
```

Inputs:  
- sSerial – The string of serialized archive.

Outputs:  
- None

Description: De-serializes a measurement archive from storage and populates the archive class.

---

**RcMeasurement Class**

Provides base class for main operations and attributes for all measurements taken within the RCCM application. Contains a reference to the control from which the measurement is being derived, a list of measurement segments, as well as information on the length and node complexity of the measurement.

<table>
<thead>
<tr>
<th>RcMeasurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Segments: ArrayList&lt;MeasurementSegment&gt;</td>
</tr>
<tr>
<td>+Control: Control</td>
</tr>
<tr>
<td>+Length: double</td>
</tr>
<tr>
<td>+Complexity: int</td>
</tr>
<tr>
<td>+RcMeasurement(Control:Control)</td>
</tr>
<tr>
<td>Measure(): void</td>
</tr>
<tr>
<td>EndMeasure(): void</td>
</tr>
</tbody>
</table>

**Requirements Satisfied**

4.4.2  4.4.6  4.4.8

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segments</td>
<td>ArrayList&lt;MeasurementSegment&gt;</td>
<td>Internal list of measurement segments that compose the composite measurement.</td>
</tr>
<tr>
<td>Control</td>
<td>Control</td>
<td>The control used to derive the measurement lengths from. This is used for establishing a relative coordinate system and calculating measurement lengths.</td>
</tr>
<tr>
<td>Length</td>
<td>double</td>
<td>The summation of the lengths of all</td>
</tr>
</tbody>
</table>
Operations

**RcMeasurement (Control Control)**

**Inputs:** Control – The Control relative to the measurement.

**Outputs:** None

**Description:** Constructs an RcMeasurement from the control on which the measurement is based.

**void Measure ()**

**Inputs:** None

**Outputs:** None

**Description:** Begins measurement process. The measurement control will be queried for measurement data until EndMeasure is called.

**void EndMeasure ()**

**Inputs:** None

**Outputs:** None

**Description:** Ends measurement process. Measurement segments will be constructed representing the total translation of the corresponding measurement control.

**RcTranslationMeasurement Class**

Derived measurement class representing zero-based, or translation measurements within the RCCM application. The measurement process uses the coordinates of an RcMotor and its subsequent translation to define measurement segments and measurement length. The measurement will “zero” coordinates on the motor at Measure(), and tally the length of the measurement based on the coordinates translation at EndMeasure().

**Requirements Satisfied**

4.4.1  4.4.6  4.4.7

**Operations**
void Measure ()
Inputs: None
Outputs: None
Description: Begins measurement process. The measurement control will be queried for its current coordinates. EndMeasure() will be called once translation is complete.

void EndMeasure ()
Inputs: None
Outputs: None
Description: Ends measurement process. Measurement segments will be constructed representing the total translation of the corresponding measurement control, relevant to its initial coordinates.

RcVectorMeasurement Class

Derived measurement class representing segment or vector based measurements in the RCCM application. The measurement process uses the coordinates of an RcMotor and its relative step length relative to the current camera view. On Measure(), points are set and segments created by the user indicating some possible course of translation of the motor. Relevant calculations are done to determine the number of motor steps needed to translate such a segment. EndMeasure returns the measurement segments defined by these calculations.

Requirements Satisfied
4.4.2  4.4.3  4.4.4  4.4.6

Operations

void Measure ()
Inputs: None
Outputs: None
Description: Begins measurement process. The measurement control will be queried for its current coordinates and relative step distance in terms of the control coordinate system. EndMeasure() will be called once measurement segments are defined on the control.

```cpp
void EndMeasure ()
Inputs: None
Outputs: None
Description: Ends measurement process. Measurement segments will be constructed based on the relative step of the motor against the control and the length of the user defined measurement segments.
```

**RcMeasurementSegment Class**

The RcMeasurementSegment class represent the measurement segments which comprise a RcMeasurement. The class provides graphic attributes for defining segment graphics in user controls, as well as begin/end coordinates and the calculated length of the segment.

```
RcMeasurementSegment
+Length: double
+Begin: RcCoordinate
+End: RcCoordinate
```

Requirements Satisfied

4.4.2

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>double</td>
<td>The length of the measurement segment.</td>
</tr>
<tr>
<td>Begin</td>
<td>RcCoordinate</td>
<td>The first coordinate of a measurement segment.</td>
</tr>
<tr>
<td>End</td>
<td>RcCoordinate</td>
<td>The second coordinate of a measurement second.</td>
</tr>
</tbody>
</table>