# F/A-18 Data Visualization Project
# CSE 498: Boeing Team – Fall 2004

Boeing Integrated Defense Systems

**Technical Specification**

Adapted from technical specifications written by previous semesters


**Group 1:**

Andrew Shulman
Jonathan Yen
David Nelson
Lun Cheung

**Table of Contents**


Introduction……. 4
Requirements and Possible Modifications..... 4
Flight Visualization Dialog …… 5
Class CFlightVisDlg…….. 6
Database Access Class……..... 13
Struct FIREntry…….. 14
Class CFIRData……. 14
Visualization Package Wrapper…….. 19
Class CVisClientsWrapper……. 22
MSU Visualization…….. 23
Class CVisClient…… 24
ActiveX Visualization Controls……. 25
Heads Up Display (HUD)……. 26
Plane visualization window……... 30
Ground Proximity Indicator (GPI)…… 35
Flight Track View (FTV)…….. 38
Missouri-Rolla Visualization…….. 42


**Figures and Tables**


Figure 1 Sequence Diagram of Data Flow from FlightVisDlg to Individual DLLs….. 4
Figure 2 FlightVisDlg State Transition Diagram……... 5
Figure 3 Global Data……..... 12
Figure 4 FIRData Table…….. 13
Figure 5 - FIRData Parameter Names……... 13
Figure 6 - Struct FIREntry…… 14
Figure 7 - Struct Function …… 19
Figure 8 Sequence Diagram of GetFunction() and ExecuteFunction() Methods…….... 20
Figure 9 - Sequence Diagram of VisClient Interactions with FlightVisDlg and DLLs... 21
Figure 10 - Sequence Diagram of MSU Visualization……. 23
Figure 11 - Sequence Diagram of Missouri-Rolla Visualization……. 42
Table 1 CFlightVisDlg Methods…… 6
Table 2 CFlightVisDlg Members…….. 10
Table 3 CFIRData Methods……... 14
Table 4 CFIRData Members ……. 17
Table 5 – CVisClientsWrapper Methods……. 22
Table 6 - CVisClientsWrapper Members…… 23
Table 7 – CVisClient Methods……. 24
Table 8 – CVisClient Members…… 24
Table 9 CHud Methods…….. 26
Table 10 – CHud Members…….. 29
Table 11 CPlane Methods…….. 30
Table 12 - CPlane Members……. 33
Table 13 CGPI Methods……. 35
Table 14 - CGPI Members…… 37
Table 15 CFlightTrackView Methods…… 38

Table 16 - CFlightTrackView Members…….. 40

**Introduction:**

The F/A-18 has an MU (Memory Unit) aboard during flights, as well as pre and post-flight. Data is collected and recorded at regular intervals, based on trigger conditions – many are recorded periodically while in flight. Types of recordings include, but are not limited to BIT (Built In Test) data, Cautions/Warnings/Advisories, discrete inputs, flight control data, engine data and maintenance codes. With an ICD (Interface Control Document) for the recorded data, various reports are generated for the different types of recordings.

This data is often used to help maintainers or engineers determine failure conditions, perform safety or mishap investigations, training and many other uses. The goal of this project is to take a related data set, parse out the data, and create several selectable displays, which visualize the data.

**This project was previously done by Capstone Classes at Michigan State University and at University of Missouri at Rolla. This project is to build on the original application, deleting certain facets and adding additional functionality.**

**Requirements:**

1. Given latitude and longitude positions, create a moving map with real terrain underneath.
2. Implement a 3D flight track showing track and altitude.
3. Create a common interface so that our visualization module works with the data feeding application from Rolla and vice versa (If Rolla is in participation)
4. It must run on Win98, NT, 2000, and XP with recommended computer specs of 1GHz with 32M 3D graphics card and 256M RAM.

**Possible Modifications:**

- Ability to start visualization at any point in the file.
- Optional start/stop time for looped playback
- **Optional frame by frame playback from a given time.**

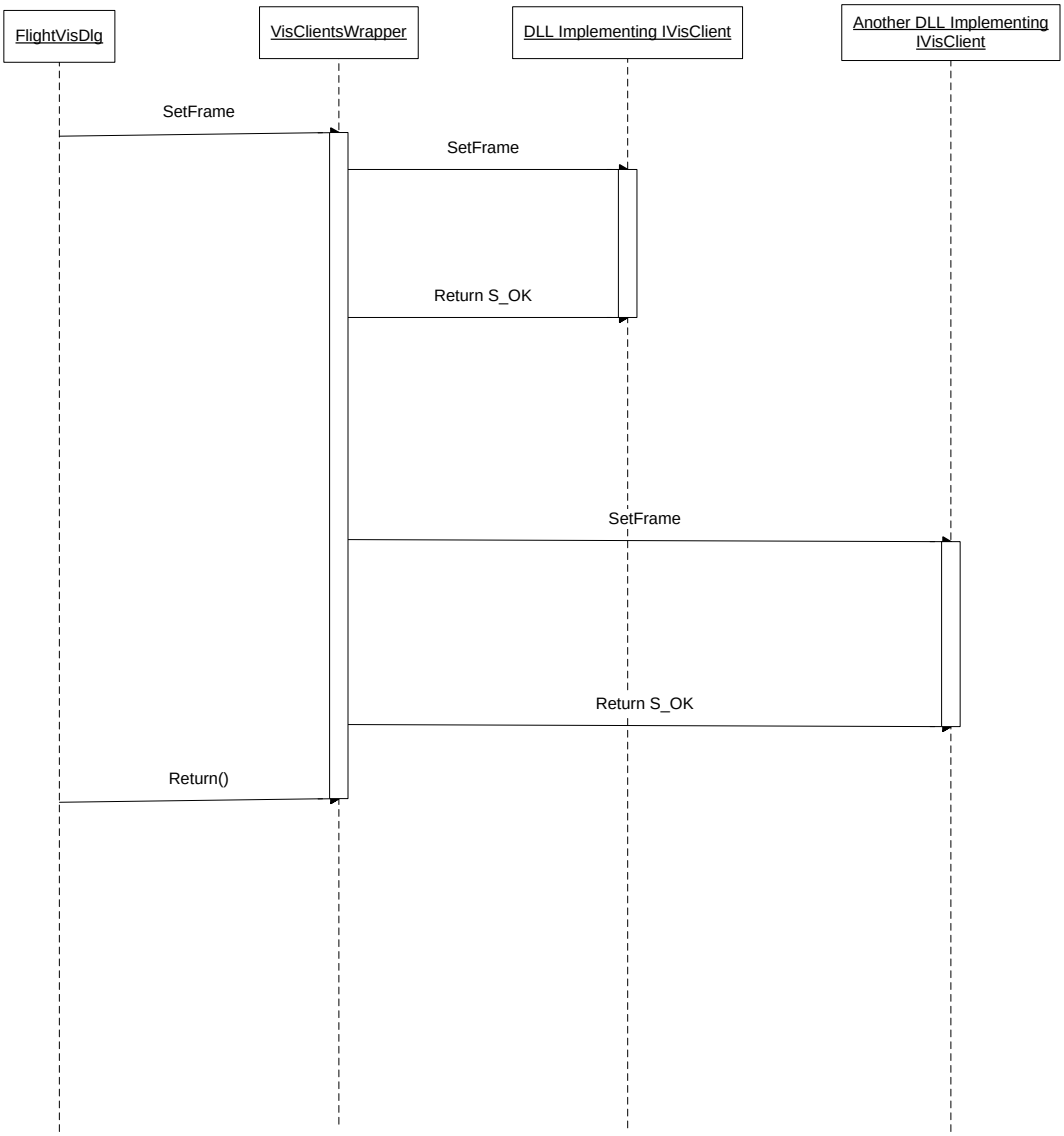


*Figure 1 – Sequence Diagram of Data Flow from FlightVisDlg to Individual DLLs*

The following technical specification details the interactions and functions of each of these classes and those specific to the MSU Flight Visualization display windows classes.

**Flight Visualization Dialog**

The Flight Visualization dialog is implemented by the CFlightVisDlg class. CFlightVisDlg holds a reference to a CFIRData object, allowing it access to the flight data. The class spawns two separate threads of execution, which run in parallel with the main thread. One is responsible for passing the information to the VisClient objects through the VisClientsWrapper objects to prompt the visualization window components to update and redraw themselves and the other is for querying the database and updating the event value buffer. The main thread handles user messages, such as mouse clicks and menu bar options. A global critical section object, CCriticalSection, is used to allow access to shared variables to the separate threads, ensuring the application is thread safe. The threads themselves are declared global since Windows threads cannot be made class members. The following diagram illustrates the basic flow of execution.

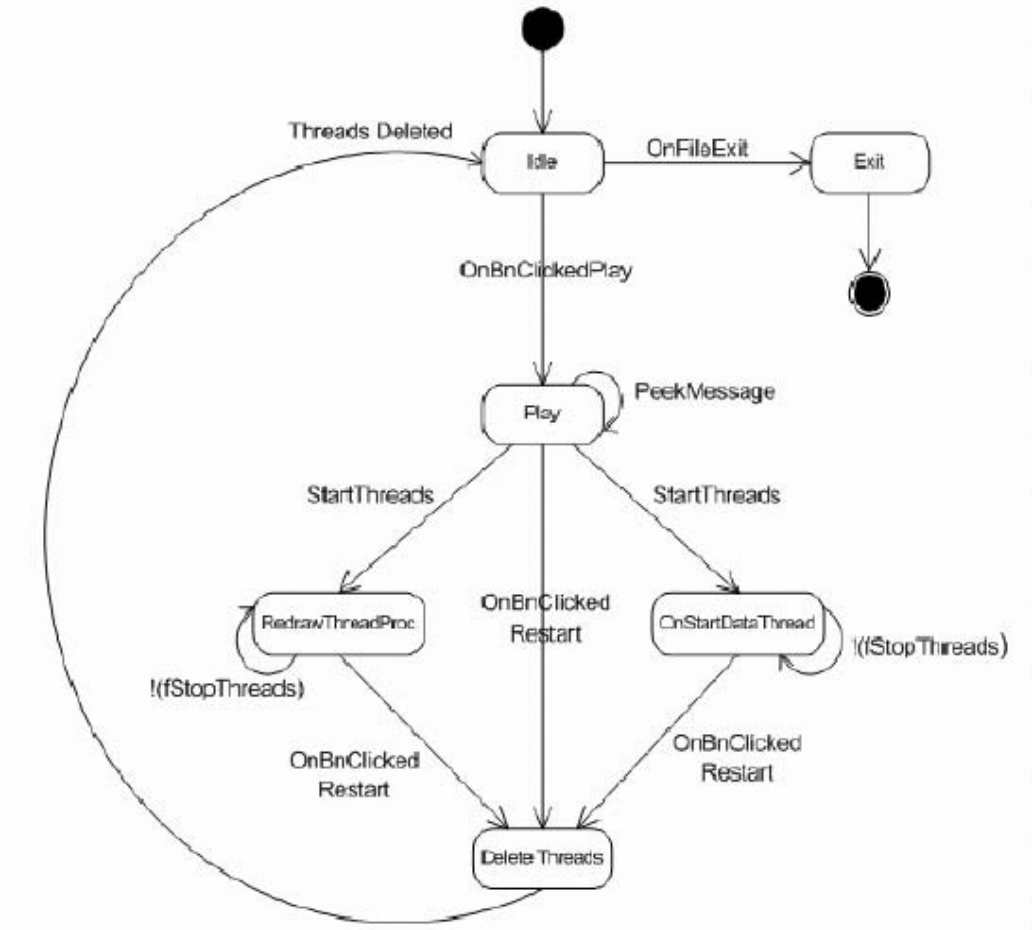


*Figure 2 – FlightVisDlg State Transition Diagram*

The class CFlightVisDlg, in addition to controlling the real-time access to the database entries, also ensures that the ActiveX controls are passed new frames of data as quickly as possible. The two worker threads created at runtime are declared OnStartDataThread and RedrawThreadProc. The basic agenda of each thread is to repeat
the following:

***OnStartDataThread***

- Obtain mutually exclusive access to the shared variables such as the CFIRData database access class, and the event value buffer, m_pEventValue.
- Obtain a buffer of data from the database. Update the event value buffer to reflect the current data values.
- Determine when the next data will be available for reading.
- Release the critical section access.
- Sleep until the next data becomes available.

***RedrawThreadProc***

- Obtain mutually exclusive access to the shared variables such as the event value buffer, m_pEventValue
- Copy the values stored in m_pEventValue locally, and release the critical section access.
- Set the frame properties of VisClientsWrapper class objects using the newly obtained, locally copied database values. VisClientsWrapper objects will in turn pass this data onto its VisClient components and their respective visualization window objects.

Each thread of execution continues this process until the variable fStopRedraw becomes true (i.e. the user presses "stop", or "pause", or the end of the database is reached). Thus, the visualization controls are redrawn at a rate which is independent of the rate of data flow. In this way, the IO-bound process of querying the database is allowed to maintain real time updates, while the compute-bound process of redrawing the OpenGL objects is allowed to function as quickly as it can. In other words, the data will always be retrieved in real time, while the graphics performance will improve as the processor and graphics hardware improve.

**Class CFlightVisDlg**

Class CFlightVisDlg is responsible for keeping the data up-to-date, and for making sure the information is passed to the VisClientsWrapper class objects at a reasonable rate. To do this, it utilizes the global worker thread functions OnStartThreads and RedrawThreadProc. This class is also responsible for responding to user mouse-clicks and button callbacks. It uses the following methods to perform these tasks:

| **Name:** | **Purpose:** | **Parameters:** | **Return:** |
|---|---|---|---|
| OnCmdMsg() | Ensures that a function executed on a visualization in the wrapper is executed only once per call | UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO* pHandlerInfo | bool |
| OnInitDialog() | Called when the dialog is initialized. Performs initialization of dialog box, buttons, bitmaps and controls such as the slider | None | bool: TRUE if successful, FALSE otherwise |
| ~CFlightVisDlg() | Destructor: deletes objects which were allocated memory by the constructor | None | None |

| | | | |
|---|---|---|---|
| CFlightVisDlg() | Constuctor: Allocates memory to pointers. | None | None |
| DisableControls() | Renders all buttons in the UI insensitive to user clicks (i.e. greys them out). | None | None |
| DoDataExchange() | Exchanges data with the user interface, placing UI defined values into member variables | CDataExchange* pDX | None |
| EnableControls() | Renders all buttons in the UI sensitive to user clicks, based upon the playback state. Certain buttons may remain grey-out. | None | None |
| OnBnClickedFastForward() | Callback for the “fast forward” button, which advances the simulator by one second during frame-by-frame playback | None | None |
| OnBnClickedFrameCheck() | Callback for the “Frame by frame” check box control, which sets the member variable m_fFrameByFrame accordingly | None | None |
| OnBnClickedLoop() | Creates a “Looped Playback” dialog box which queries the user for a looping range. When the dialog is closed, looping is automatically enabled. | None | None |
| OnBnClickedLoopCheck() | Sets the check state of the “Looped Playback” check box, and sets the member variable m_fLooping accordingly | None | None |
| OnBnClickedNextTrack() | Callback for the “next track” button, which advances the flight simulator to the next flight. | None | None |
| OnBnClickedPause() | Sets the playback stat to PAUSE, and gStopRedraw to TRUE, thereby halting the threads | None | None |
| OnBnClickedPlay() | Sets the playback state to PLAY and begins the | None | None |

| | | | |
|---|---|---|---|
| | worker threads OnStartDataThread and RedrawThreadProc | | |
| OnBnClickedPreviousTrack() | Callback for the “previous track” button, which moves the flight simulator to the previous flight in a multi-flight database. | None | None |
| OnBnClickedRestart() | Sets the playback state to STOP, and gStopRedraw to TRUE. Also sets the slider position to 0, and moves to the beginning of the database | None | None |
| OnBnClickedRewind() | Callback for the “rewind” button, which moves the simulator back by one second during frame-by-frame playback. | None | None |
| OnBnClickedSlowdown() | Callback for the “decrease playback speed” button. Decreases the value of m_PlaybackSpeed | None | None |
| OnBnClickedSpeedup() | Callback for the “increase playback speed” button. Increases the value of m_PlaybackSpeed | None | None |
| OnCancel() | Calls OnFileExit(). Used when closing window by “Cancel Window” button | None | None |
| OnCBNComboSelChange() | Callback for the combo box control. This function is called when the selection in the combo box is changed | None | None |
| OnFileExit() | Stops the threads, closes the database and exits the application | None | None |
| OnFileOpendata() | Initializes the database file for reading, determines the number of flights in the database, and allows the user to select which flight to display. | None | None |
| OnHelpGethelp() | Opens HTML help files in a web browser | None | None |

| | | | |
|---|---|---|---|
| OnHScroll() | Callback for the slider control, when it is moved, and when the thumb is let go | UINT nSBCode: the reason for the call.<br>UINT nPos: The new slider position.<br>CScrollBar* pSlider: Pointer to the control. | None |
| OnPaint() | Called by the system to redraw the application dialog when it is repositioned or resized | None | None |
| OnQueryDragIcon() | The framework calls this function by a minimized (iconic) window that does not have an icon defined for the class. | None | None |
| OnSysCommand() | The framework calls this member function when the user selects a command from the Control menu, or when the user selects the Maximize or Minimize button. | UINT: specifies the type of system command requested.<br>LPARAM: cursor coordinates | None |
| StartThreads() | Starts the worker threads, sets gStopRedraw to FALSE, and sets up a message dispatch to listen for user messages | None | None |

*Table 1 CFlightVisDlg Methods*

**Data Members:**

Additionally, CFlightVisDlg utilizes the following member variables:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_DBFile | _bstr_t | The name of the Microsoft Access database file to open, as supplied by the user |
| m_fFrameByFrame | bool | TRUE if frame by frame playback is enabled, FALSE otherwise. This flag causes frame by frame to start or stop. |
| m_fLooping | bool | TRUE if looped playback is enable, FALSE otherwise. This flag causes looping to start or stop |
| m_fStopThreads | bool | TRUE if the threads should not be running, FALSE otherwise. This flag |

| | | |
|---|---|---|
| | | causes the separate threads to stop running |
| once | bool | TRUE if function has been run once |
| m_stop | bool | TRUE if the threads should be reset, FALSE otherwise |
| m_cbFrame | CButton | Check box used to indicate whether or not frame by frame is enabled |
| m_cbLoop | CButton | Check box used to indicate whether or not looping is enabled |
| m_clCombo | CComboBox | The class that encapsulates a combo box object, used to display the different flights within a multi-flight database and to allow the user to switch flights after loading the data |
| m_Database | CFIRData | A database object, which allows access to the database values via a recordset pointer. |
| m_pcwd | char* | Pointer to the semi current working directory |
| m_FastForwardButton | CImageButton | Used to place a bitmap over the dialog fast forward button, rather that the standard text |
| m_LoopButton | CImageButton | Used to place a bitmap over the dialog loop button, rather that the standard text |
| m_NextTrackButton | CImageButton | Used to place a bitmap over the dialog next track button, rather that the standard text |
| m_PauseButton | CImageButton | Used to place a bitmap over the dialog pause button, rather that the standard text |
| m_PlayButton | CImageButton | Used to place a bitmap over the dialog play button, rather that the standard text |
| m_PreviousTrackButton | CImageButton | Used to place a bitmap over the dialog previous track button, rather that the standard text |
| m_RewindButton | CImageButton | Used to place a bitmap over the dialog rewind button, rather that the standard text |
| m_SlowdownButton | CImageButton | Used to place a bitmap over the dialog slowdown button, rather that the standard text |
| m_SpeedupButton | CImageButton | Used to place a bitmap over the dialog speedup button, rather that the standard text |
| m_StopButton | CImageButton | Used to place a bitmap over the dialog stop button, rather that the standard text |
| m_odtLoopStart | COleDateTime | The start time of the looping range |
| m_odtLoopStop | COleDateTime | The end time of the looping range |
| m_pControls | ControlContainer* | A structure containing pointers to the |

| | | |
|---|---|---|
| | | three ActiveX control objects. This struct is passed to the RedrawThreadProc, which uses them to redraw the controls |
| m_clSlider | CSliderCtrl | The class that encapsulates a slider object, used to show the current reading position in the database, relative to the beginning and end |
| m_PlaybackSpeedLabel | Cstring | The static text drawn into the “Playback Speed” field of the dialog box |
| m_sAbsoluteTime | Cstring | The time displayed by the TimeLabel control, used to keep track of the exact time to be displayed after pausing, or using frame-by-frame playback |
| m_TimeLabel | Cstring | The static text drawn into the “Time” field of the dialog box. |
| m_pEventValue | double* | A pointer to an array of double precision digits, which represent the values in the datatbase at any given point in time after the OnStartDataThread has begun |
| m_PlayBackState | enum PlaybackState | The current state of the playback state machine, which can be one of: PLAY, PAUSE, STOP |
| m_nCurrFlight | int | The current flight being played, based on the number of flights in the database |
| m_nFrameDir | int | The direction of playback, requested by the user during frame-by-frame playback. A value of 0 indicates forward; 1 indicates backward |
| m_nPlaybackSpeed | int | The playback speed, which can be one of: 1, 2, 5, 10, 15, 20, or 25 |
| m_nSliderStart | int | The starting position of th slider during looped playback |
| m_nSliderStop | int | The ending position of the slider during looped playback |
| m_hIcon | HICON | Application icon |
| m_pVisClients | CVisClientsWrapper* | Pointer to the visualization wrapper |
| m_SliderPos | int | The position of the slider, relative to the beginning. The range of positions is set in OnInitDialog |

*Table 2 CFlightVisDlg Members*

**Global Data:**

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| g_dttime | COleDateTime | Timestamp of entry |
| g_millisecond | int | Milliseconds of entry |

*Figure 3 – Global Data*

**Database Access Class**

The flight data is supplied to the visualization tool in a Microsoft Access database. The database contains a single table called FIRData. The FIRData table contains four separate fields comprising all of the data necessary to run the visualization tool: DtTime, Milliseconds, ParameterName and ParameterValue.

| **FIRData** |
|---|
| DtTime<br>Milliseconds<br>ParameterName<br>ParameterValue |

*Figure 4 – FIRData*

The DtTime field contains a string consisting of the date and time that the associated parameter was recorded, in the form: mm/dd/yyyy hh:mm:ss PM. The table is accessed through the C++ class CFIRData. The Milliseconds field consists of the number of milliseconds after the time specified by the associated DtTime field at which the associated parameter was recorded. It is an unsigned integer. The ParameterName field consists of a string that specifies the parameter being recorded at the given time. Possible parameter names include:

| **Name:** | **Meaning:** |
|---|---|
| Airspeed | Airspeed, in knots |
| Altitude | Altitude of the plane, in feet |
| AOA | The angle of attack of the plane |
| EWV | The east-west velocity |
| G | The G-Force felt by the pilot |
| GEARUP | The state of the landing gear |
| Heading | Magnetic heading, in degrees |
| LANDING | The plane is considered as landed |
| NSV | The north-south velocity |
| Pitch | Angle of the plane with its latitudinal axis |
| PitchRate | Rate of change of the pitch |
| Roll | Angle of the plane with its longitudinal axis |
| RollRate | Rate of change of the roll |
| TAKEOFF | The plane is considered to have taken off |
| VV | The vertical velocity |
| WOW | The weight on wheels of the plane |
| YawRate | Rate of change of the yaw (angle with respect to the plane's vertical axis). |
| NLAT | Latitude |
| ELONG | Longitude |

*Figure 5 - FIRData Parameter Names*

Finally, the ParameterValue field contains a floating point digit that represents the value of one of the parameter names described above.

**Struct FIREntry**

The FIREntry struct is used as a means of conveniently storing all of the information contained in a single record. Since the entries in the ParameterName field are enumerable, they are stored as an enumeration to save space. FIREntry consists of the following fields:

| FIREntry |
|---|
| enum EventType |
| COleDateTime dttime |
| double parametervalue |
| EventType parametername |
| int recordNum |
| UINT milliseconds |

*Figure 6 - Struct FIREntry*

**Class CFIRData**

The CFIRData class utilizes Microsoft ADO 2.7 (ActiveX Data Objects version 2.7) to access the database fields. A connection to the database is established using an ADO _ConnectionPtr object, which is then used to generate a _RecordsetPtr pointer object via an SQL SELECT statement. This object is used to advance through the records, one by one. It cannot be used to move backwards through the records, but this is not problematic since the SQL SELECT can be written such that the records are received in sorted order. The methods supplied by the CFIRData class include**:**

| **Name:** | **Purpose:** | **Parameters:** | **Return Value:** |
|---|---|---|---|
| ~CFIRData() | Destructor | None | None |
| ACKSeek() | Sets the member variable m_fSeeked to FALSE, and causes the function Seeked() to return FALSE | None | None |
| CFIRData() | Constructor | None | CFIRData object |
| CloseFlightData() | Closes the connection established with InitializeDatabase(), and calls Reset() | None | None |
| GetBuffer() | Queries the database for a DtTime value which matches the current date and time stored in a member variable. All entries for the given second are sorted by milliseconds, and store din a FIREntry struct. Each struct is inserted | None<br>or<br>int n_desiredSeconds | vector<struct FIREntry>&:<br>Reference to the vector containing the FIREntry's for the current second |

| | | | |
|---|---|---|---|
| | into a vector of FIREntry's , and is returned to the caller | | |
| GetCurr_RecordNum() | Returns the index of the current record retrieved from the database | None | long: the index of the current record retrieved from the database |
| GetDtTime() | Returns the current DtTime field from whatever database record is currently being pointed to | None | CString: The DtTime value as a string |
| GetFlightDate() | Returns the date of the flight specified by its index | int flight_num | Cstring: the date of the flight as a string |
| GetFlightEnd() | Returns the ending time of the flight specified by its index | int flight_num | Cstring: the time of the flight as a string |
| GetNumFlights() | Returns the number of flights in the database | None | int: the number of flights in the database |
| GetNumRecords() | Returns the total number of records in the database | None | long: the number of records in the database |
| GetRecordByDate() | Returns the record number of the last occurrence of the specified date | CString sDate | long: the record number of the last occurrence of the date specified |
| InitializeDatabase() | Opens the database named by its parameter. Initializes the connection pointer and recordset pointer to the beginning of the dtatbase | const_bstr_t filename | None |
| IsEnd() | Returns TRUE if the database reader has reached the end of the file | None | VARIANT_BOOL: TRUE if the database reader has reached the end of the file |
| IsOpen() | Returns TRUE if a flight database is currently open. Returns FALSE otherwise | None | bool: TRUE if the database is open |
| OpenFlightData() | Scans the database opened with InitializeDatabase() for multiple flights, and stores the total number of records per flight. | None | None |

| | | | |
|---|---|---|---|
| | Also, stores the values for min/max airspeed and altitude | | |
| ReportException() | Instantiates a popup dialog box containing the reason the exception was thrown. Waits for the user to acknowledge, then exits the application | char* reason, _com_error ex | None |
| Reset() | Clears all flight record and min/max values from local variables. Useful when closing a database connection | None | None |
| Seek() | Seeks through the database until the supplied record is reached. The recordset pointer is then set to point to this position | Long nNum | long: the record number being pointed at, or -1 if failure |
| Seeked() | Returns TRUE if the function Seek() has been called and the function ACKSeek() has not been called. Returns FALSE otherwise | None | bool: TRUE if the database has been Seeked and not ACKED |
| SetFlight() | Sets the member variable m_odtCurrTime to the time specified by the caller | int nFlightNum | None |
| SetTime() | Sets the member variable m_odtCurrTime to the time specified by the caller | UINT nHour<br>UINT nMinute<br>UINT nSecond | None |
| TimeDifference() | Calculates the time difference in milliseconds between two times | ColeDateTime t1, t2<br>UINT m1, m2 | double: the difference in time |

*Table 3 CFIRData Methods*

**Member Data:**

Class CFIRData also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_Connection | _ConnectionPtr | Points to a connection with a |

| | | |
|---|---|---|
| | | Microsoft Access database. The connection is then used to generate recordset pointers from SQL queries |
| m_ConnectRead | _RecordsetPtr | Points to a recordset object. The pointer always points to the current record that is being examined |
| m_FirstRead | bool | TRUE if the current pass through the database is the first |
| m_fOpen | bool | TRUE if a connection pointer to a database has been opened. FALSE otherwise |
| m_fSeeked | bool | TRUE if the database has been Seeked() without being ACKed() |
| m_odtCurrDate | COleDateTime | The current date and time used to fill the m_vCurrSecond buffer. It is incremented by 1 second each time a new buffer is filled. |
| m_odtNextDate | COleDateTime | The date that should be used as the ending date to accept while performing GetBuffer(). |
| m_odtsDesiredSpan | COleDateTimeSpan | The span of desired seconds passed to GetBuffer(). This is the number of seconds worth of data (real time) that will be passed to the caller. |
| m_nCurrFlight | int | The current flight in a multiple flight database |
| m_nFirstRecordNumber | Int | The absolute number of the first record of the selected flight, relative to the absolute beginning of the database. |
| m_nCurrRecord | long int | The current record number pointed to by m_ConnectRead, relative to the first record of that flight |
| m_nNumFlights | int | The number of different flights in the database |
| m_nNumRecords | long int | The number of records in the flight being examined |
| m_vFlightDates | vector<CString> | Contains the starting DtTime value of each flight tin the database. The user chooses the desired flight based on this value |
| m_vFlightEnds | vector<CString> | Contains the ending DtTime value of each flight in the database. Useful for knowing the limits of a looping range |
| m_vCurrSecond | vector<FIREntry> | Contains all of the data from the database that matches the SQL query |

| | | |
|---|---|---|
| | | for the current DtTime value, sorted by milliseconds |
| m_vNumFlightRecs | vector<long> | Contains the number of records for each flight within the database being examined. The vector can be indexed by the desired flight number |
| m_vRecordMap | vector<map<CString, int>> | Contains the map of the first occurrence of each DtTime value, mapped to the record number of that occurrence. Useful in the GetDateByRecord, and frame by frame playback. |

*Table 4 CFIRData Members*

**Visualization Package Wrapper - CVisClientsWrapper**

This class acts as a wrapper for multiple visualization DLLs to facilitate simultaneous playback. The CVisClient Wrapper object keeps a vector of pointers to visualizations selected for playback as well as a vector of Function structs containing information about the functions associated with the selected visualizations. The purpose of the Function vector is to enable calls to functions that are defined within the DLL and need to be called as the result of user input, but are not available through the IVisClients interface. The Function struct consists of the following fields:

| **Function** |
|---|
| char Name[255]<br>int WrapperIndex<br>int VisIndex<br>int FunctIndex |

*Figure 7 - Struct Function*

Name contains the name of the function. WrapperIndex and VisIndex provide the indices to of the wrapper and visualization with which the function is associated. FunctIndex is the index of the function within its visualization.

The wrapper constructor calls the method GetFunctions() on itself which calls a similar method on all its visualization DLLs. The functions return a VARIANT type, this type is used so that the dlls can still be used be Visual Basic. The VARIANT holds a SAFEARRAY of BSTR which each contain the name of one function. The order of the function names is assumed to also

be the order of their indexes in the dll, starting at 0 (i.e. if the array contained {"Open Model", "Hide HUD", "Hide GPI"} then ExecuteFunctions(0) would open a new model, 1 would hide the HUD, etc.). The BSTR objects are converted to char[255] when placed into a Function struct, so no conversion is needed when using that data type. The wrapper uses the method ExecuteFunction() to execute functions associated with a DLL. When ExecuteFunction(wIndex) is called, a call is made to m_visualization[vIndex]->ExecuteFunction(fIndex), where wIndex is the index of the function in the m_functions vector, vIndex is the index of the visualization that the function belongs to in the m_visualizations vector, and fIndex is the index inside the visualization of the function. The following sequence diagram illustrates the use of these functions:

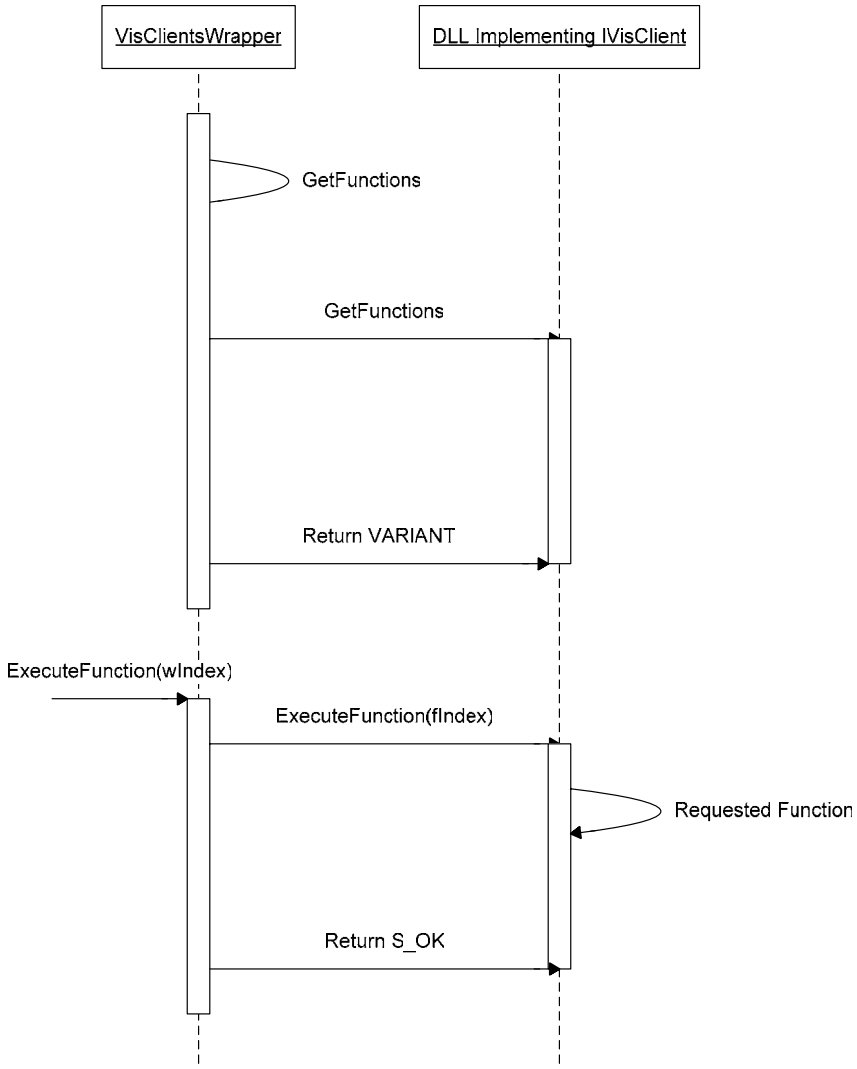


*Figure 8 – Sequence Diagram of GetFunction() and ExecuteFunction() Methods*

The following sequence diagram illustrates the interaction of the CVisClientsWrapper class with the FlightVisDlg and its VisClient DLLs, where CVisClientsWrapper is the .dll wrapper class.

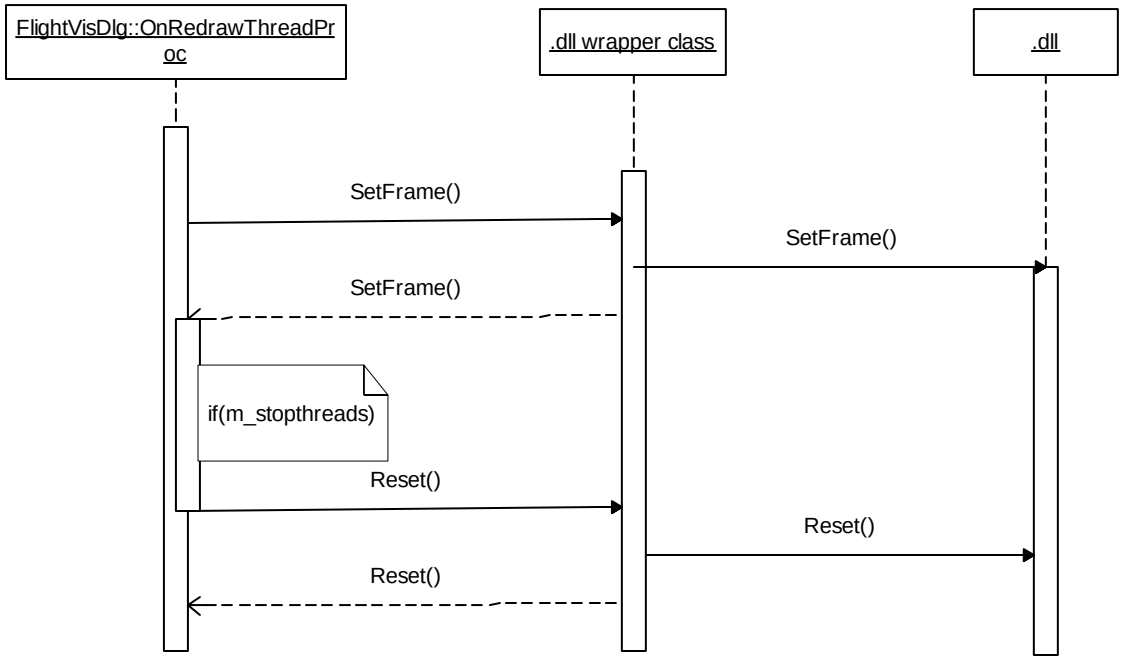


*Figure 9 – Sequence Diagram of VisClient Interactions with FlightVsDlg and DLLs*

**Class CVisClientsWrapper**

VisClientsWrapper has the following methods to receive information from the FlightVisDlg and pass it onto its VisClient DLLs:

| **Name:** | **Purpose:** | **Parameters:** | **Return:** |
|---|---|---|---|
| CVisClientsWrapper() | Constructor. Creates any IVisClient pointers for any visualization DLLs to be used and adds them to the m_visualizations vector | None | None |
| ~CVisClientsWrapper() | Destructor | None | None |
| Reset() | Calls Reset() on all visualizations in m_visualizations vector. This will return all variables in a visualization to their initial values. Usually used after a pause/stop | None | None |
| SetFrame() | Sets the data for this frame in all visualizations | FIRStruct* f, FIRTimeStamp* g | None |
| GetNumberOfVisualizations() | Returns the number of visualizations being handled by this wrapper | None | size_t |
| GetDescriptionOf() | Returns the description of the Visualization at the integer index specified in the parameters. This information is stored in the DLL and retrieved by the GetName() method of IVisClient interface | unsigned index | char* |
| GetFunctionsFor() | Retrieves all functions from the visualization that corresponds to the vIndex passed | unsigned vIndex | vector<Function> |
| ExecuteFunction() | Executes the function corresponding to the wIndex passed | unsigned wIndex | bool |
| GetFunctions() | Retrieves all function info from the DLLs and places them into the m_functions vector. This method should be called after all visualizations have been added (ie: after the constructor). | None | None |

*Table 5 - CVisClientsWrapper Methods*

**Member Data:**

Class CVisClientsWrapper also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_visualizations | vector<IVisClientPtr> | Vector containing pointers to selected |

| | | |
|---|---|---|
| | | visualization DLLs |
| m_functions | vector<Function> | Vector containing Function structs for the selected visualizations |

*Table 6 - CVisClientsWrapper Members*

**MSU Visualization - CVisClient**

This class instantiates and contains the window components of the MSU Flight Visualization—the Plane visualization, the HUD, the Flight Track View, and the Ground Proximity Indicator. CVisClient passes the flight data received from FlightVisDlg by way of the VisClientsWrapper class to the individual window components objects using its SetFrame() method, which calls the SetFrame() method of the window components. This process is illustrated in the sequence diagram below:

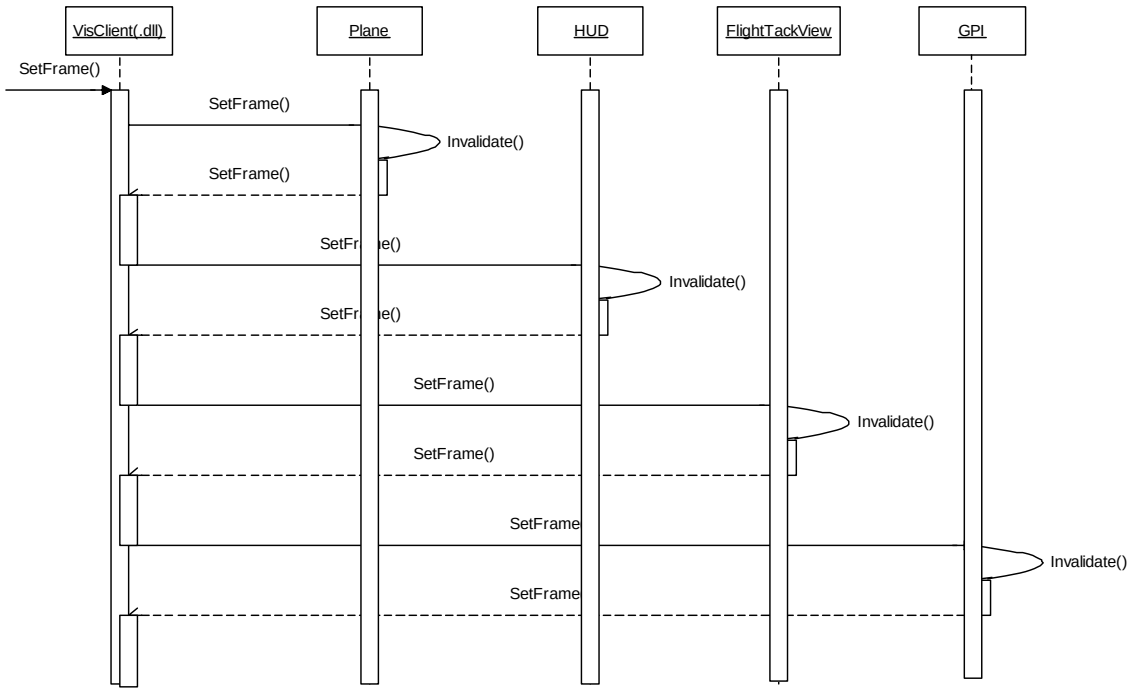


*Figure 10 – Sequence Diagram of MSU Visualization*

**Class CVisClient**

CVisClient uses the following methods to accept data from the CVisClientsWrapper class and pass it to its four window component objects:

| **Name:** | **Purpose:** | **Parameters:** | **Return:** |
|---|---|---|---|
| CVisClient() | Constructor for CVisClient | None | None |

| | | | |
|---|---|---|---|
| | object. Instantiates each of the MSU Flight Visualization window objects | | |
| ~CVisClient() | Destructor | None | None |
| FinalConstruct() | Called by the implementation when the object has finished constructing itself | None | HRESULT |
| Reset() | Sets m_curtime to NULL. Calls Reset() or clear() function on all window objects | None | None |
| SetFrame() | Calls SetFrame() function on all window objects. Calculates position variables m_curx, m_cury, m_curz | FIRStruct *f, FIRTimeStamp* g | None |
| GetName() | Returns string identifying object as MSU Flight Visualization | BSTR *name | None |
| GetFunction() | Retrieve available functions for selected visualization | VARIANT * pVal | None |
| ExecuteFunction() | Execute available functions for selected visualization | int fNum | None |

*Table 7 - CVisClient Methods*

**Member Data:**

Class CVisClient also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_GPI | CGPI* | Pointer to GPI window object |
| m_ HUD | CHUD* | Pointer to HUD window object |
| m_plane | CPlane* | Pointer to plane window object |
| m_ftv | CFlightTrackView* | Pointer to FlightTrackView window object |
| m_name | char* | Pointer to memory containing name of visualization |
| m_curx | double | Stores current east-west position |
| m_ cury | double | Stores current north-south position |
| m_curz | double | Stores velocity vector information |
| m_curtime | COleDateTime | Stores date and time of current second being processed |
| m_curmillisec | int | Current millisecond within second being processed |

*Table 8 - CVisClient Members*

**ActiveX Visualization Controls:**

The Active X visualization controls were created using the Active Template Library 7.0 and OpenGL libraries. They each implement all standard interfaces defined by the Microsoft Visual Studio .NET wizard and also allow connection points with their container(s). They are all contained within the amudvt project workspace, which is an ATL based project without MFC support. Each control functions along the same basic principles.

Upon creation, from the OnCreate function, the control will initialize the OpenGL context in which it will draw, using functions like CreateContext and CreateRGBPalette. Then, anytime the programmer/user wishes to have the control redraw itself with new parameter values, all that must be done is to call SetFrame and the will redraw itself. See the following section, "Tips for Reusing Visualizations," for more information on how to use the controls within your own separate executable. All drawing is actually done in the OnDraw function.

The controls can also be resized easily since all sizes are computed from the controls' bounding rectangle rather than from absolute pixel values. Upon notification of a size message, the control simple destroys its old OpenGL context and recreates one with the proper size.

**Tips for Reusing Visualizations:**

The visualizations can be reused in any other application of your choosing in order to allow future users/programmers to feed in data from different sources besides a pre-created Microsoft Access database. They can be inserted easily into any Visual Studio and/or .net application. Granted, you will need a basic knowledge of how to use standard controls in visual studio, which would be too in depth to detail here. Then, each time SetFrame is called, the visualization(s) will update themselves. These methods are described in the grids below.

Note that the visualization controls will update whenever those methods are invoked, so keep in mind that you must watch their timing. In general, if ran on a computer with the recommended specifications, they should be able to approach twenty-eight frames per second maximum. As the capabilities of the computer grow, so too will the frame rate. When ran in the provided stand-alone Flight Visualization application, the data will update at a fixed maximum of ten millisecond intervals due to the potential delay in SQL queries on the database.

**HUD (ActiveX Control class for the HUD)**

This class displays the Heads Up Display in an ActiveX control.

| Name: | Purpose: | Parameters: | Return: |
|---|---|---|---|
| ~CHUD() | Destructor deletes all heap memory used | None | None |
| bSetupPixelFormat() | Uses a PIXELFORMATDESCRIPTOR struct to setup the pixel format used in CreateContext(). More specifically, the pixel format is RGBA type with 24-bit color depth. It is double buffered with a 32-bit z buffer. | HDC hdc | bool: indicates success of setup |
| CHUD() | Constructor creates defaultPalEntry struct for use to later initialize the color palette for OpenGl context. Also sets all members to necessary initialization values such as setting pointers to NULL parameters to zero | None | None |
| ComponentFromIndex() | Used by CreateRGBPalette to optimally compute color entries through table lookups rather than actual calculations. Improves speed of CreateContext, and thereby allows for faster resizing when CreateContext is called by OnSize | int i,<br>UINT nbits,<br>UINT shift | unsigned char |
| CreateContext() | Sets up the OpenGL context in which the visualization is drawn in. Sets up the pixel format through a call to CreateRGBpalette() and attaches the RGB color palette created in the constructor to the context. Also initializes the camera perspective for the OpenGL function calls | HDC hdc,<br>RECT& rc | None |
| CreateRGBPalette() | Called by CreateContext() to initialize the color palette using the default palette entry struct value | HDC hdc | None |
| DrawAirSpeedAndAltitude( ) | Displays airspeed and altitude inside boxes | double speed,<br>double altitude | None |

| | | | |
|---|---|---|---|
| DrawGandAOA() | Displays G's and AOA | double G, double AOA | None |
| DrawHSI() | Draws heading scale indicator | double heading | None |
| DrawHSITics() | Draws tic marks for heading scale indicator | double height, double width, double Heading | None |
| DrawLatLon() | Calculates latitude and longitude and draws latitude/longitude display | double NSV, double EWV, double altitude, double latitude, double longitude | None |
| DrawLine() | Draws a single line of the pitch ladder and displays the number associated with that line | int num | None |
| DrawPitchLadder() | Makes all calls to draw the pitch ladder | double FPA, double PLBank | None |
| DrawRollInd() | Draws roll indicator | double roll | None |
| DrawRollTic() | Draws one individual tic mark for roll indicator | double depth | None |
| DrawVelocityVector() | Makes all calls to draw the velocity vector | double NSV, double EWV, double VV, double Roll, double Pitch, double &velX, double &velY, bool lowVelocity | None |
| FinalConstruct() | Called by the implementation when the object has finished constructing itself | None | HRESULT |
| FinalRelease() | Called by the implementation when the object is about to destroy itself because there are no extant references left on the object | None | None |
| OnCreate() | Called after construction, upon creating of control. Saves the handle to the device context into a member variable (m_hdc), determines the bounding rectangle, and initializes the control by calling CreateContext(). | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnDestroy() | Deletes the OpenGL context and sets the m_hdc to NULL | UINT uMsg, WPARAM wParam, LPARAM lParam, | LRESULT |

| | | | |
|---|---|---|---|
| | | BOOL& bHandled | |
| OnDraw() | Sets the current OpenGL context to the one created by this control in case it has been changed since the last call, then sets up OpenGL parameters, calculates data variables, and makes calls to draw actual visualization. | ATL_DRAWINFO & di | HRESULT |
| OnEraseBackgnd() | Returns zero to eliminate flicker. Stop the erasure of background. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnFirstDraw() | Creates font display list | None | None |
| OnSize() | After ensuring that the current OpenGL context is the one created by this class's CreateContext function, it deletes the now invalid context and recreates it to occupy the new size of the control's border rectangle. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| Reset() | Resets HUD to a starting state | None | None |
| SetFrame() | Control method to set all property values in one call | VisClientStruct* f | None |

*Table 9 CHud Methods*

**Member Data:**

Class HUD also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_bActive | bool | TRUE if mouse is active |
| m_bMouseCaptured | bool | TRUE if mouse position is captured |
| m_firstdraw | bool | TRUE if the first frame has not been drawn |
| m_Airspeed | double | Stores property value |
| m_Altitude | double | Stores property value |
| m_AOA | double | Stores property value |
| m_EWV | double | Stores property value |
| m_G | double | Stores property value |
| m_Heading | double | Stores property value |
| m_Lat | double | Stores calculated latitude value |
| m_Lon | double | Stores calculated longitude value |
| m_NSV | double | Stores property value |
| m_Pitch | double | Stores property value |
| m_Roll | double | Stores property value |

| | | |
|---|---|---|
| m_VV | double | Stores property value |
| m_Fov | float | Field of view for HUD |
| m_fRadius | float | Window radius |
| m_base | GLunit | Font display list |
| m_hdc | HDC | Stores handle to control's GDI device context |
| m_hrc | HGLRC | Window context |
| m_hPal | HPALETTE | Palette context |
| defaultOverride[13] | int | |
| m_fontHeight | int | Height of font for HUD |
| m_xPos | int | Mouse position x-coord |
| m_yPos | int | Mouse position y-coord |
| *m_pPal | LOGPALETTE | Log context |
| defaultPalEntry[20] | PALETTEENTRY | Stores default palette color values, used by CreateRGBPalette |
| oneto8[2] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| threeto8[8] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| twoto8[4] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| m_NLAT | double | Stores property value |
| m_ELONG | double | Stores property value |

*Table 10 CHud Members*

**Plane (ActiveX Control class for the Plane)**

This class displays the actual model of the selected aircraft image in an ActiveX control.

| **Name:** | **Purpose:** | **Parameters:** | **Return:** |
|---|---|---|---|
| ~CPlane() | destructor deletes all heap memory used | None | None |
| bSetupPixelFormat() | Uses a PIXELFORMATDESCRIPTOR struct to setup the pixel format used in CreateContext(). More specifically, the pixel format is RGBA type with 24-bit color depth. It is double buffered with a 32 bit z buffer. | HDC hdc | bool: indicates success of setup |
| ComponentFromIndex() | used by CreateRGBPalette to optimally compute color entries through table lookups rather than actual calculations. Improves speed of | int i,<br>UINT nbits,<br>UINT shift | unsigned char |

| | | | |
|---|---|---|---|
| | CreateContext, and thereby allows for faster resizing when CreateContext is called by OnSize | | |
| Cplane() | constructor creates defaultPalEntry struct for use to later initialize the color palette for OpenGl context. Also sets all members to necessary initialization values such as NULL-ing pointers and zeroing out visualization parameters. | None | None |
| CreateContext() | sets up the OpenGL context in which the visualization is drawn in. Sets up the pixel format through a call to CreateRGBpalette() and attaches the RGB color palette created in the constructor to the context. Also initializes the camera perspective for the OpenGL function calls | HDC hdc, RECT& rc | None |
| CreateRGBPalette() | called by CreateContext() to initialize the color palette using the default palette entry struct value | HDC hdc | None |
| CreateTexture() | Creates Texture information for texture mapping of OpenGL object | GLunit textureArray[], LPSTR strFileName, int textureID | None |
| DrawFrontGear() | Draws front gear | None | None |
| DrawLandingGear() | Draws gear set | None | None |
| DrawRearGear() | Draws rear gear | None | None |
| DrawRunway() | Contains the OpenGL code to draw the runway at the location given | Double x, y, z | None |
| FinalConstruct() | Called by the implementation when the object has finished constructing itself | | HRESULT |
| FinalRelease() | Called by the implementation when the object is about to destroy itself because there are no extant references left on the object | None | None |
| needsRedraw() | Returns value of m_redraw | none | bool: indicates |

| | | | |
|---|---|---|---|
| | | | if redraw needed |
| OnCreate() | called after construction, upon creating of control. Saves the handle to the device context into a member variable (m_hdc), determines the bounding rectangle, and initializes the control by calling CreateContext(). | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnDestroy() | deletes the OpenGL context and sets the m_hdc to NULL | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnDraw() | Sets the current OpenGL context to the one created by this control in case it has been changed since the last call, then sets up OpenGL parameters, calculates data variables, and makes calls to draw actual visualization. | ATL_DRAWINFO & di | HRESULT |
| OnEraseBackgnd() | returns zero to eliminate flicker. Stop the erasure of background. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnFirstDraw() | Creates bitmap font display list on the first pass | None | None |
| OnSize() | after ensuring that the current OpenGL context is the one created by this class's CreateContext function, it deletes the now invalid context and recreates it to occupy the new size of the control's border rectangle. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OpenModel() | Opens the 3DS model pointed to by m_Filename | None | None |
| Plane() | Draws the plane when called by OnDraw | GLdouble& x_extent, GLdouble& y_extent, GLdouble& z_extent | GLdouble |

| | | | |
|---|---|---|---|
| put_Filename() | Sets filename of plane model and opens model | char* newVal | None |
| Reset() | Resets Plane back to starting state | None | None |
| setFilename() | Sets filename of plane model, opens model, and redraws | LPSTR filename | None |
| SetFrame() | Control method to set all property values in one call | VisClientStruct* f | None |
| setNeedsRedraw() | Sets m_redraw to TRUE | None | None |
| UpdateGear() | Updates gear | None | None |
| DrawBuilding() | Draw building object | double x, double y, double z, int tex | None |

*Table 11 - CPlane Methods*

**Member Data:**

Class Plane also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| MAX_DEPTH | #define | Sets limits for plane |
| MAX_HEIGHT | #define | Sets limits for plane |
| MAX_WIDTH | #define | Sets limits for plane |
| m_firstdraw | bool | Check for first draw |
| m_isNewModel | bool | Check for newly loaded model |
| m_redraw | bool | Indicates whether the window needs to be redrawn |
| time | bool | Check for initial pass |
| m_Loader | CLoad3DS | 3D model loader object |
| m_alt | double | Current altitude of the plane |
| m_endx | double | Stores the final x coordinate of the plane |
| m_endy | double | Stores the final y coordinate of the plane |
| m_endz | double | Stores the final z coordinate of the plane |
| m_final_alt | double | Stores the final altitude of the current flight |
| m_GearDown | double | Stores property value |
| m_Heading | double | Stores property value |
| m_init_alt | double | Stores the initial altitude of the current flight |
| m_landed | double | Stores property value for LANDING |
| m_Pitch | double | Stores property value |
| m_Roll | double | Stores property value |
| m_speed | double | Current speed of the plane |
| m_x | double | Stores the current x coordinate of the plane |

| | | |
|---|---|---|
| m_y | double | Stores the current y coordinate of the plane |
| m_z | double | Stores the current z coordinate of the plane |
| t_gearstate {RAISING, LOWERING} | enum | Stores status of the landing gear |
| m_planeDL | GLunit | Plane display list |
| m_Textures[MAX_TEXTURES] | GLunit | Texture object for model |
| m_runwayTex[MAX_TEXTURES] | GLunit | Texture object for runway |
| m_hdc | HDC | Stores handle of control's device context |
| m_hrc | HGLRC | Window context |
| m_hPal | HPALETTE | Palette context |
| defaultOverride[13] | int | |
| dist | int | Distance traveled along runway |
| m_initialDraw | int | Indicates if is initial draw |
| m_GearState | int | Numeric value of landing gear status |
| *m_pPal | LOGPALETTE | log context |
| defaultPalEntry[20] | PALETTEENTRY | stores default palette color values, used by CreateRGBPalette |
| m_Filename | string | Filename of 3DS model being used |
| m_3DModel | t3dModel | Loaded 3D model |
| oneto8[2] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| threeto8[8] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| twoto8[4] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |

*Table 12 - CPlane Members*

**CGpi (ActiveX Control class for the GPI)**

This class displays the Ground Proximity Indicator in an ActiveX control.

| **Name:** | **Purpose:** | **Parameters:** | **Return:** |
|---|---|---|---|
| bSetupPixelFormat() | Uses a PIXELFORMATDESCRIPTOR struct to setup the pixel format used in CreateContext(). More specifically, the pixel format is RGBA type with 24-bit color depth. It is double buffered with a 32 bit z buffer. | HDC hdc | bool: indicates success of setup |
| FinalConstruct() | Called by the implementation when the object has finished constructing itself | None | HRESULT |
| OnDraw() | Sets the current OpenGL context to the one created by this control in case it has been changed since the last call, then sets up OpenGL parameters, calculates data variables, and makes calls to draw actual visualization. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | HRESULT |
| OnCreate() | Called after construction, upon creating of control. Saves the handle to the device context into a member variable (m_hdc), determines the bounding rectangle, and initializes the control by calling CreateContext(). | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnDestroy() | Deletes the OpenGL context and sets the m_hdc to NULL | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnEraseBackgnd() | Returns zero to eliminate flicker. Stop the erasure of background. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnSize() | After ensuring that the current | UINT uMsg, | LRESULT |

| | | | |
|---|---|---|---|
| | OpenGL context is the one created by this class's CreateContext function, it deletes the now invalid context and recreates it to occupy the new size of the control's border rectangle. | WPARAM wParam, LPARAM lParam, BOOL& bHandled | |
| CreateContext() | Sets up the OpenGL context in which the visualization is drawn in. Sets up the pixel format through a call to CreateRGBpalette() and attaches the RGB color palette created in the constructor to the context. Also initializes the camera perspective for the OpenGL function calls | HDC hdc, RECT& rc | None |
| CreateRGBPalette() | Called by CreateContext() to initialize the color palette using the default palette entry struct value | HDC hdc | None |
| DisplayInteger() | Draws an integer number at the specified window coordinates | int num, double x=0, double y=0, double z=0 | None |
| DrawAltTics() | Draws tic marks for ground proximity indicator | None | None |
| DrawGround() | Draws the ground at m_takeoffAlt | None | None |
| FinalRelease() | Called by the implementation when the object is about to destroy itself because there are no extant references left on the object | None | None |
| OnFirstDraw() | Creates bitmap font display list on the first pass | None | None |
| Reset() | Resets window back to starting state | None | None |
| SetFrame() | Control method to set all property values in one call | VisClientStruct* f | None |
| ComponentFromIndex() | used by CreateRGBPalette to optimally compute color entries through table lookups rather than actual calculations. Improves speed of CreateContext, and thereby allows for faster resizing when CreateContext is called by | int i, UINT nbits, UINT shift | unsigned char |

| | | | |
|---|---|---|---|
| | OnSize | | |

*Table 13 CGPI Methods*

**Member Data:**

Class CGpi also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_firstdraw | bool | TRUE if first time drawing |
| m_Altitude | double | Stores property value |
| m_maxAlt | double | Stores maximum altitude of the flight |
| m_minAlt | double | Stores minimum altitude of the flight |
| m_takeoffAlt | double | Stores altitude at takeoff |
| m_base | GLuint | Font display list |
| m_hdc | HDC | GPI device context |
| m_fontHeight | int | Height of font |
| m_pixelsize | double | |
| m_fRadius | float | Window radius |
| m_hPal | HPALETTE | Palette context |
| defaultOverride[13] | int | |
| *m_pPal | LOGPALETTE | Log Context |
| defaultPalEntry[20] | PALETTEENTRY | Stores default palette color values, used by CreateRGBPalette |
| oneto8[2] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| threeto8[8] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| twoto8[4] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |

*Table 14 CGPI Members*

**CFlightTrackView (ActiveX Control class for the FTV)**

This class displays the Flight Track View in an ActiveX control.

| **Name:** | **Purpose:** | **Parameters:** | **Return:** |
|---|---|---|---|
| bSetupPixelFormat() | Uses a | HDC hdc | bool: indicates |

| | | | |
|---|---|---|---|
| | PIXELFORMATDESCRIPTOR struct to setup the pixel format used in CreateContext(). More specifically, the pixel format is RGBA type with 24-bit color depth. It is double buffered with a 32 bit z buffer. | | success of setup |
| FinalConstruct() | Called by the implementation when the object has finished constructing itself | None | HRESULT |
| OnDraw() | Sets the current OpenGL context to the one created by this control in case it has been changed since the last call, then sets up OpenGL parameters, calculates data variables, and makes calls to draw actual visualization. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | HRESULT |
| OnCreate() | Called after construction, upon creating of control. Saves the handle to the device context into a member variable (m_hdc), determines the bounding rectangle, and initializes the control by calling CreateContext(). | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnDestroy() | Deletes the OpenGL context and sets the m_hdc to NULL | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnEraseBackgnd() | Returns zero to eliminate flicker. Stop the erasure of background. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| OnSize() | After ensuring that the current OpenGL context is the one created by this class's CreateContext function, it deletes the now invalid context and recreates it to occupy the new size of the control's border rectangle. | UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled | LRESULT |
| CreateContext() | Sets up the OpenGL context | HDC hdc, | None |

| | | | |
|---|---|---|---|
| | in which the visualization is drawn in. Sets up the pixel format through a call to CreateRGBpalette() and attaches the RGB color palette created in the constructor to the context. Also initializes the camera perspective for the OpenGL function calls | RECT& rc | |
| CreateRGBPalette() | Called by CreateContext() to initialize the color palette using the default palette entry struct value | HDC hdc | None |
| FinalRelease() | Called by the implementation when the object is about to destroy itself because there are no extant references left on the object | None | None |
| OnFirstDraw() | Creates bitmap font display list on the first pass | None | None |
| SetFrame() | Control method to set all property values in one call | VisClientStruct* f | None |
| Scales() | Rescales window | None | None |
| clear() | Clears window, initializes window | None | None |
| ComponentFromIndex() | used by CreateRGBPalette to optimally compute color entries through table lookups rather than actual calculations. Improves speed of CreateContext, and thereby allows for faster resizing when CreateContext is called by OnSize | int i,<br>UINT nbits,<br>UINT shift | unsigned char |
| LoadBitMapFile() | Load a bitmap file | char* filename,<br>BITMAPINFOHEADER* bitmapInfoHeader | unsigned char* |
| DrawTextureMap() | Draw the map | None | None |

*Table 15 CFlightTrackView Methods*

**Member Data:**

Class CFlightTrackView also consists of the following data members:

| **Name:** | **Data Type:** | **Description:** |
|---|---|---|
| m_firstdraw | bool | TRUE if first time drawing |

| | | |
|---|---|---|
| m_heading | double | Stores property value |
| m_x | double | X-coordinate of current location |
| m_z | double | Z-coordinate of current location |
| m_x_min | double | Minimum X-coordinate |
| m_z_min | double | Maximum Z-coordinate |
| m_x_max | double | Minimum X-coordinate |
| m_z_max | double | Maximum Z-coordinate |
| m_xSCALE | double | Scale in the X-direction |
| m_zSCALE | double | Scale in the Z-direction |
| m_xCENTER | double | Center of X-direction range |
| m_zCENTER | double | Center of Y-direction range |
| m_Fov | float | Field of view for FTV |
| m_base | GLuint | Font display list |
| m_hdc | HDC | GPI device context |
| m_hrc | HGLRC | Window context |
| m_fontHeight | int | Height of font |
| dot | struct | Struct holding coordinates of point to be drawn in window |
| m_dots | std::vector<dot> | Vector of dots to display |
| m_fRadius | float | Window radius |
| m_hPal | HPALETTE | Palette context |
| defaultOverride[13] | int | |
| *m_pPal | LOGPALETTE | Log Context |
| defaultPalEntry[20] | PALETTEENTRY | Stores default palette color values, used by CreateRGBPalette |
| oneto8[2] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| threeto8[8] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| twoto8[4] | unsigned char | Used as mathematical table look up for ComponentFromIndex() |
| m_map_min_x | double | Minimum X-coordinate for map |
| m_map_min_z | double | Minimum Z-coordinate for map |
| m_map_max_x | double | Maximum X-coordinate for map |
| m_map_max_z | double | Maximum Z-coordinate for map |

*Table 16 CFlightTrackView Members*

**Rolla Visualization**

The Rolla2MSU DLL converts calls made to the MSU interface into those that Rolla's dll can use. Thus, when it receives a SetFrame call Rolla2MSU takes all the information out of the parameters passed to it and calls UpdateFlightDataRaw. UpdateFlightDataRaw takes about 22 parameters, a convenience function that acts as a workaround for interface problems with their struct.

The following sequence diagram illustrates the interaction between the VisClientsWrapper, the Rolla2MSU visualization, and its visualization client DLLs.

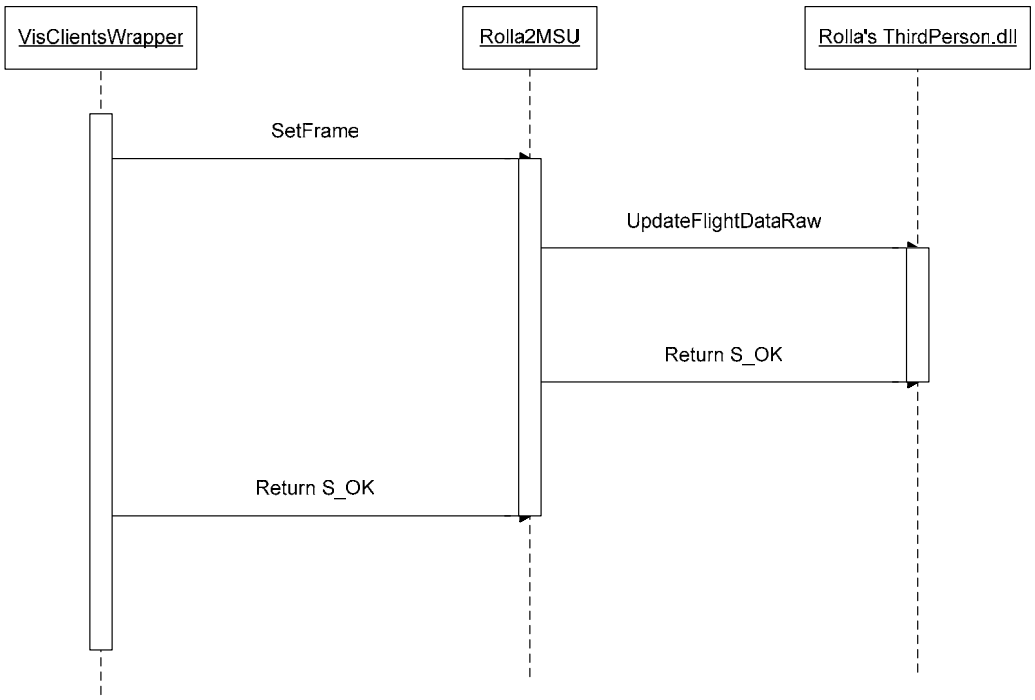


*Figure 11 – Sequence Diagram of Missouri-Rolla Visualization*