Real-time Display Driver

MDS
AERO SUPPORT CORPORATION

Proprietary Notice

TABLE OF CONTENTS

1. REAL-TIME DISPLAY DRIVER

1.1 Introduction

1.1.1 ProDAS (Professional Data Acquisition System) is a data acquisition system for gas turbine test cells.

1.1.2 The RTD Driver is part of the RTD System of proDAS, it is used to display pages and spotlight sticky display windows during a test run. This RTD System also comprises the Real-time Display ActiveX Controls and the RTD Editor (cf. Engineering Specification of Real-time Display ActiveX Controls and Engineering Specification of Real-time Display Editor, respectively)

1.1.3 This document defines the COM interfaces to be used by the clients of the RTD Driver.

1.2 Purpose

1.2.1 Using the interfaces of the RTD Driver, clients will be able to control the display and selection of display pages, creation of spotlight sticky display windows, and specification of fullsets to be displayed in appropriate Real-time Display ActiveX Controls.

1.2.2 Clients known up to now are the Management GUI and test procedures, i.e. automation clients using VBS, as well as other instances of the RTD Driver which may request displaying fullsets.

1.2.3 On any single computer, only a single instance of the RTD Driver will run. Typically, the automation clients of the RTD Driver reside on another computer than the RTD Driver being controlled. For this remote control, DCOM will be used.

1.3 Scope

1.3.1 This document is intended for programmers of the COM clients (e.g. Management GUI and test procedures) using the COM interfaces specified herein as well as the programmers of the RTD Driver offering these COM interfaces.

1.4 Applicable Documents

Document number	Description
ES78001.2620	Functional Requirements Document for ProDAS

Document number	Description
ES78022.2766	Engineering Specification of Real-time Display ActiveX Controls
ES78021.2650	Engineering Specification of Real-time Display Editor
ES78022.2651	Engineering Specification of Real-time Display Driver
ICD78022.2802	Interface Control Document of Real-time Display ActiveX Controls

1.5 Codes and Standards

N/A

1.6 Abbreviations and Definitions

Term	Description
BMP	File extension for bitmap files
C++	Programming language
ClassID	Uniquely identifies the RTD Driver from other ActiveX controls. The control identifier is the registry key number and is the same for every instance of the RTD Driver that is used.
CoClass	Component Object Class (the class implementing an interface)
COM	Component Object Model
DCOM	Distributed Component Object Model
DLL	Dynamic Link Library
EMF	File extension for files containing an image in Enhanced Metafile Format

Term	**Description**
EXE	File extension for executable files
FIFO	First in – first out (access scheme for queues)
GUI	Graphical User Interface
ICD	Interface Control Document
IDL	Interface Definition Language
MTA	Multithreaded Apartment
OCX	File extension for ActiveX Controls (formerly called OLE Custom Controls)
ProgID	The ProgID representing the object. ProgIDs present a human-readable version of the class identifier (ClassID) used to identify COM/ActiveX objects.
RTDRoot	Root directory of a directory tree containing the files with display pages. Immediately below the RTDRoot directory there are the engine type specific directories.
RTE	Real Time Engine
STA	Single-Threaded Apartment
VB	Programming language Visual Basic
VBS	Visual Basic Script

2. DESIGN

2.1 Introduction

2.1.1 The RTD Driver is a DCOM server, i.e. it can be used over the network.

2.1.2 The RTD Driver is an out-of process server.

2.1.3 The RTD Driver has the file extension “exe”.

2.1.4 The RTD Driver is an executable.

2.1.5 There may be several computers on which the RTD Driver is running at the same time.

2.1.6 On each of these computers only a single instance of the RTD Driver will be able to run at the same time.

2.1.7 Each client may access and control one or more instances of the RTD Driver at the same time.

2.1.8 Each instance of the RTD Driver may be accessed and controlled by several clients as well as by an interactive user. However, it is assumed that these clients co-operate and that no precautionary measures are required to ensure this.

2.1.9 The clients may reside on the same computer as the RTD Driver.

2.1.10 The RTD Driver is implemented as a MTA server. It is thread-safe, i.e. critical code sections will be safeguarded against simultaneous access by several clients. This involves the interactive user as well as automation clients. However, all clients will share the same objects and data of the RTD Driver, and since subsequent method calls froms different clients could disturb each other it will be up to the clients to co-operate responsible.

2.1.11 Each view and each spotlight sticky display window runs in an own thread of the RTD Driver. If two clients access the same view at the same time this might lead to unpredictable results.

2.1.12 The RTD Driver shall not terminate automatically but only on explicit request by an interactive user or a client using the method *IApplication::Terminate*.

2.1.13 The same holds for views and spotlight sticky display windows: They shall not be closed automatically when the associated interface has been released but on explicit request only using the appropriate *Close* method.

2.1.14 The RTD Driver does not offer any other interfaces besides the interfaces described in this document.

2.1.15 The RTD Driver shall be self-registering by calling it with the argument "/RegServer". Registration will take place when proDAS is installed. The RTD Driver will have to be resident and installed as well on all server computers (e.g. display cmoputers) as on all client computers (e.g. User Interface PC). On server computers, following the registration of the RTD Driver, the appropriate rights should be set using the Windows utility *dcomcnfg*. These privileges should comprise starting and accessing the RTD Driver by dedicated users from remote client computers.

2.1.16 The appropriate users will be able to start the RTD Driver on any of the server computers and they will be able to access and control the single instance of the RTD Driver which may already be running on a server computer by adding the name of the server computer in their *CreateObject* call, e.g. VBS clients should connect with

```
Set RTDD = CreateObject (“proDAS.RTDDriver”, “PC1234”)
```

2.1.17 Since there will only be a single instance of the RTD Driver running simultaneously on any server computer, the client will be automatically connected to this instance instead of launching an additional instance.

2.1.18 The method *LastCallHasFailed* of the interface *IErrors* (cf. 2.6.3.1) may be used to determine whether any method or property call has failed. Whenever a call fails it will generate an error message that can be retrieved using the interface IErrors (for an example see 2.6.7.1). Such error situations are marked “Error:” in the following descriptions of methods and properties.

2.1.19 The RTD Driver offers five interfaces. Their methods and properties as well as their relationship are presented in Figure 1. Note that this figure does not actually display the interfaces but the corresponding implementation classes (CoClasses) since the aggregation relations cannot be displayed for interfaces but for classes only. Nevertheless, the figure should provide an overview of the interfaces. The correspondence between the CoClasses and interfaces should be clear: Replacing the initial “C” by an “I” and removing the trailing “Impl” gives the interface name.

2.1.20 This document contains a number of Visual Basic Script examples illustrating the interfaces. Although these examples are working scripts, they are by no means complete Visual Basic Scripts but concentrate on the functionality of the RTD Driver. For instance, a VBS programmer would be well advised to enforce explicit declaration of variables.

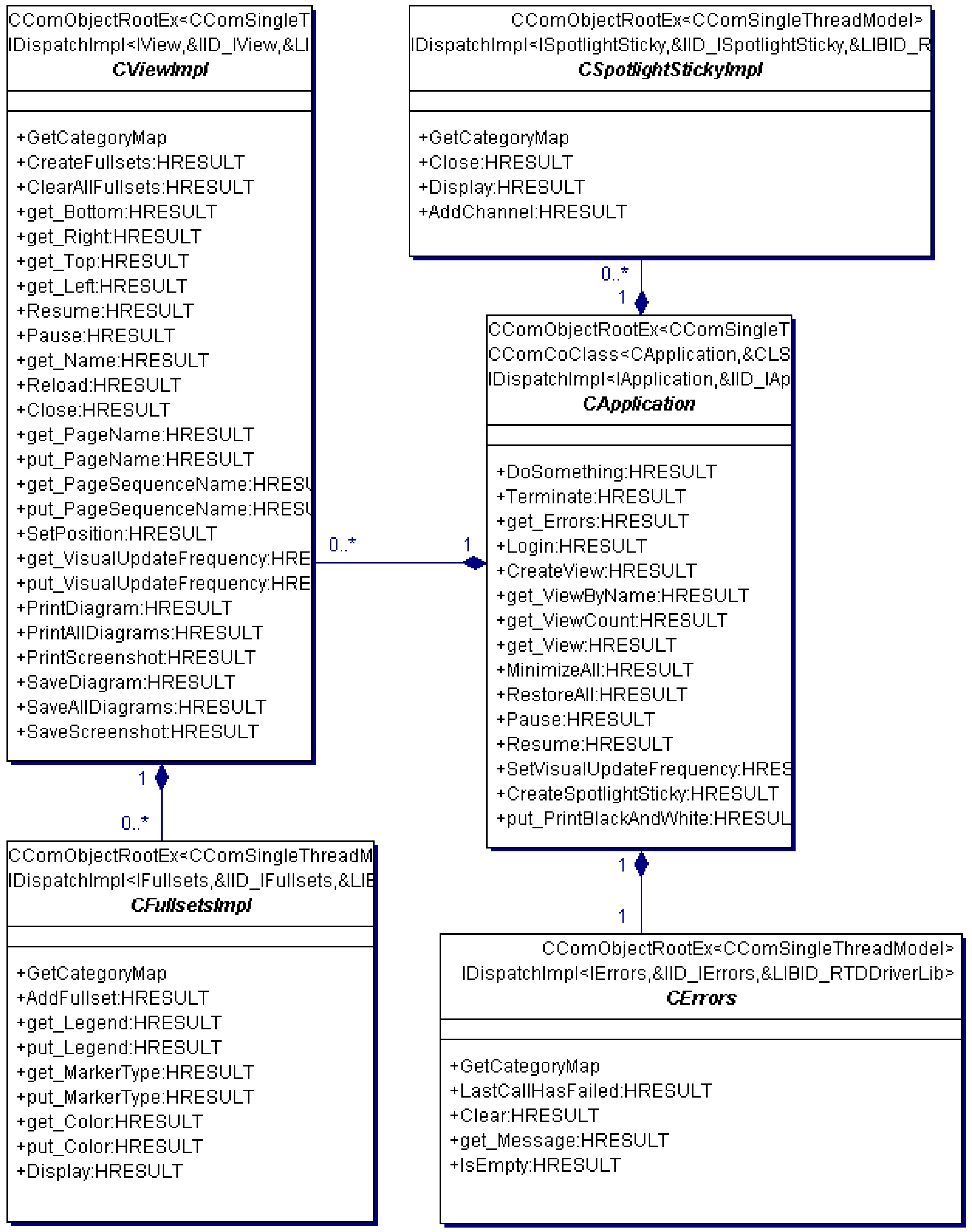

Figure 1: CoClasses

2.2 Interface *IApplication*

2.2.1 General

2.2.1.1 The interface IApplication helps control the RTD Driver application.

2.2.1.2 The RTD Driver runs only once on a single computer.

2.2.1.3 The ProgID of this interface is “proDAS.RTDDriver".

2.2.1.4 This interface provides the starting point for any client. All the other interfaces defined in this document will be accessed from here; their objects cannot be created directly.

2.2.2 Design

2.2.2.1 This is a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.2.3 Methods and Properties

2.2.3.1 Method *Login*

```
[id(1)] HRESULT Login([in] BSTR username, [in] BSTR password,
            [out, retval] BOOL * success)
```

Argument Name	Description
`username`	The name of the user, case-sensitive.
`password`	The password of the user, case-sensitive.
`success`	If login was successful, `success` is TRUE, else FALSE.

2.2.3.1.1 Using this method, a proDAS user may login to the RTD Driver.

2.2.3.2 Method *CreateView*

```
[id(2)] HRESULT CreateView([in] BSTR name, [out, retval] IView**
            pView)
```

Argument Name	Description
`name`	The unique name of the new view or an empty string indicating a default name is to be assigned by the RTD Driver.
`pView`	points to an object implementing the interface *IView* (cf. 2.3)

2.2.3.2.1 This method creates a new named view and returns the associated interface pointer.

2.2.3.2.2 Error: If a view with the given name already exists, a default name will be assigned to the newly created view and an error message will be generated (cf. 2.6.3.3).

2.2.3.3 Property *ViewByName*

```
[propget, id(3)] HRESULT ViewByName([in] BSTR name, [out, retval]
            IView** pView)
```

Argument Name	Description
`Name`	The name of the view, case-insensitive.
`pView`	`pView` points to the interface *IView* associated to the view with the given name. If no such view exists, `NULL` is returned.

2.2.3.3.1 This property returns the interface pointer associated to the view with the given name.

2.2.3.4 Property *ViewCount*

```
[propget, id(4)] HRESULT ViewCount([out, retval] long *pVal))
```

Argument Name	Description
`pVal`	The number of views currently displayed (regardless if minimized or visible).

2.2.3.4.1 This property returns the number of currently displayed views.

2.2.3.5 Property *View*

```
[propget, id(5)] HRESULT View([in] long lIndex, [out, retval]
             IView** pView)
```

Argument Name	Description
`lIndex`	One based index of a view. This index refers to the collection of all currently displayed views. Therefore, `lIndex` must not exceed the total number of currently displayed views as returned by the property `ViewCount.`
`pView`	`pView` points to the interface *IView* associated to the requested view. If no such view exists, i.e. the index is out of its valid range, `NULL` is returned.

2.2.3.5.1 This property returns the interface pointer associated to the view with the given index.

2.2.3.6 Method *MinimizeAll*

```
[id(6)] HRESULT MinimizeAll()
```

This method has no arguments.

2.2.3.6.1 This method minimises all windows of the RTD Driver, i.e. its main window, all view windows, and all spotlight sticky display windows.

2.2.3.6.2 If all RTD Driver windows are already minimised, the method has no effect.

2.2.3.6.3 Note that minimised views containing Real-time Display ActiveX Controls with buffered data will continue retrieving data while all other views as well as spotlight sticky display windows will pause data retrieval when minimised.

2.2.3.7 Method *RestoreAll*

```
[id(7)] HRESULT RestoreAll()
```

This method has no arguments.

2.2.3.7.1 This method restores all previously minimised windows of the RTD Driver. If all of its windows are visible already, it has no effect.

2.2.3.8 Method *Pause*

```
[id(8)] HRESULT Pause()
```

This method has no arguments.

2.2.3.8.1 This method pauses data display for all views. Note that spotlight sticky display windows will not pause. If all views already are in pause state, this method has no effect.

2.2.3.9 Method *Resume*

```
[id(9)] HRESULT Resume()
```

This method has no arguments.

2.2.3.9.1 This method resumes data display for all previously paused views. If no view is in pause state, this method has no effect.

2.2.3.10 Method *SetVisualUpdateFrequency*

```
[id(10)] HRESULT SetVisualUpdateFrequency([in] float newVal)
```

Argument Name	**Description**
`newVal`	New value of the visual update frequency in Hz.

2.2.3.10.1 This method sets the visual update frequency for all current and future views and spotlight sticky display windows, respectively.

2.2.3.10.2 Error: If `newVal` exceeds the maximum value of 10 Hz, this maximum value will be set and an error message will be generated (cf. 2.6.3.3).

2.2.3.11 Method *CreateSpotlightSticky*

```
[id(11)] HRESULT CreateSpotlightSticky([out, retval]
            ISpotlightSticky** pSpotlightSticky)
```

Argument Name	**Description**
`pSpotlightSticky`	points to the object implementing the interface *ISpotlightSticky* (cf. 2.4) of the newly created spotlight sticky display window.

2.2.3.11.1 This method creates a new spotlight sticky display window and returns the associated interface pointer.

2.2.3.12 Property *PrintBlackAndWhite*

```
[propput, id(12)] HRESULT PrintBlackAndWhite([in] BOOL
          BlackAndWhite)
```

Argument Name	Description
`BlackAndWhite`	Indicates whether the printer shall be handled as a black and white printer.

2.2.3.12.1 This property enables the client to specify whether the printer shall be handled as a black and white printer when printing diagrams.

2.2.3.13 Method *Terminate*

```
[id(13)] HRESULT Terminate([in] BOOL bForce)
```

Argument Name	Description
`bForce`	Forces the RTD Driver to terminate even if other clients are still connected to it.

2.2.3.13.1 This method terminates the application, i.e. all currently displayed views and spotlight sticky display windows will be closed, and the RTD Driver will terminate.

2.2.3.13.2 If there are other clients still connected to the RTD Driver apart from an interactive user, it will not terminate unless `bForce` is TRUE.

2.2.3.14 Property *Errors*

```
[propget, id(14)] HRESULT Errors([out, retval] IErrors** pError)
```

Argument Name	Description
`pError`	Points to the object implementing the interface IErrors (cf. 2.5.7.1) associated with the single error object of the RTD Driver.

2.2.3.14.1 This property gives access to the single error object of the current connection (i.e. *IApplication* object) to the RTD Driver, thus allowing the user to retrieve his own error messages.

2.2.4 Events Fired

2.2.4.1 This interface has no events.

2.2.5 Usage Conditions and Restrictions

2.2.5.1 All calls must occur between *Login* and *Terminate*.

2.2.5.2 Calling the (get) property *Errors* several times does not help (but does not do any harm either), it merely retrieves always the same object.

2.2.5.3 The property *ViewByName* should not be used before a view with the given name has been created, otherwise NULL will be returned.

2.2.5.4 The property *View* should not be useed with an index less than one or greater than the value returned by *ViewCount*, otherwise NULL would be returned.

2.2.5.5 An example of a valid calling sequence is given subsequently. Note that calls within the same line (separated by a hyphen) may occur in any sequence:

- Login
- Errors
- CreateView
- get ViewCount
- get ViewByName – get View
- PrintBlackAndWhite
- Pause
- MinimizeAll
- SetVisualUpdateFrequency
- Resume
- CreateSpotlightSticky
- RestoreAll
- Terminate

2.2.5.6 The RTD Driver shall not terminate automatically when the interface has been released but only on explicit request by an interactive user or a client using the method *Terminate*.

2.2.6 Persistent Data

2.2.6.1 The Engineering Specification of Real-time Display Driver describes which data are persistently stored in the initialisation file of the RTD Driver.

2.2.7 *EXAMPLE*

2.2.7.1 The following example illustrates the usage of the interface *IApplication*:

```
set RTDD = CreateObject ("proDAS.RTDDriver", “MPC6464”)
RTDD.Login "Thomas", "mySecretPassword"
set Errors = RTDD.Errors
set View1 = RTDD.CreateView("test1", error)
RTDD.CreateView "test2"
RTDD.CreateView "test3"
msgbox "viewname = " + View1.Name      ‘ =”test1”
nViews = RTDD.ViewCount     ‘ = 3
RTDD.Pause
set View2 = RTDD.View(2)    ‘ = the view named “test2”
set View3 = RTDD.ViewByName("test3")
RTDD.PrintBlackAndWhite = TRUE
RTDD.SetVisualUpdateFrequency 5
set SLS1 = RTDD.CreateSpotlightSticky
set SLS2 = RTDD.CreateSpotlightSticky
RTDD.Resume
HasTerminated = RTDD.Terminate(TRUE)   ‘ enforce termination
```

2.3 Interface *IView*

2.3.1 *General*

2.3.1.1 The interface *IView* represents a single view window and can be used to control the page selection and the appearance of the view.

2.3.1.2 An object for this interface cannot be created directly but through the method IApplication::CreateView only.

2.3.2 *Design*

2.3.2.1 This is a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.3.3 *Methods and Properties*

2.3.3.1 Property *Name*

```
[propget, id(1)] HRESULT Name([out, retval] BSTR *pVal)
```

Argument Name	Description
`pVal`	Name of the view.

2.3.3.1.1 This property returns the name of the view.

2.3.3.2 Method *Close*

```
[id(2)] HRESULT Close()
```

This method has no arguments.

2.3.3.2.1 This method closes the view window and deletes the associated object. Consequently, the pointer to the view becomes invalid and must not be used thereafter.

2.3.3.3 Property *PageSequenceName*

```
[propget, id(3)] HRESULT PageSequenceName([out, retval] BSTR
            *pVal)

[propput, id(3)] HRESULT PageSequenceName([in] BSTR newVal)
```

Argument Name	**Description**
`pVal, newVal`	Name of a page sequence stored in the Sequence.xml file and defined using the RTD Editor. When input, `pVal` is interpreted case-insensitively.

2.3.3.3.1 This property enables the user to set a new current page sequence and to retrieve the current setting.

2.3.3.4 Property *PageName*

```
[propget, id(4)] HRESULT PageName([out, retval] BSTR *pVal)

[propput, id(4)] HRESULT PageName([in] BSTR newVal)
```

Argument Name	**Description**
`pVal, newVal`	Path name of a file containing a display page. This may be an absolute path name starting with “<drive>:\” or “\\” and a relative path name which will be interpreted relative to the current engine type subdirectory of the RTDRoot directory. Normally, views are stored in files with the extension ‘v’, but all other extensions are supported as well. When input, `pVal` is interpreted case-insensitively.

2.3.3.4.1 This property enables the user to read and display a display page as well as to retrieve the file name of the currently displayed page.

2.3.3.5 Method *Reload*

```
[id(5)] HRESULT Reload()
```

This method has no arguments.

2.3.3.5.1 This method reloads the currently displayed page, i.e. the view is read again and the display page is re-initialised. This method is most helpful when the display page has been modified using the RTD Editor and shall reflect these modifications.

2.3.3.6 Method *SetPosition*

```
[id(6)] HRESULT SetPosition([in] long left, [in] long top, [in]
          long right, [in] long bottom)
```

Argument Name	Description
`Left`	Pixel co-ordinate of the left window border or -1 to indicate that the current value is not to be changed.
`Top`	Pixel co-ordinate of the upper window border or -1 to indicate that the current value is not to be changed.
`Right`	Pixel co-ordinate of the right window border or -1 to indicate that the current value is not to be changed.
`Bottom`	Pixel co-ordinate of the lower window border or -1 to indicate that the current value is not to be changed.

This method sets a new position for the view window.

2.3.3.7 Property *Left*

```
[propget, id(7)] HRESULT Left([out, retval] long *pVal)
```

Argument Name	Description
`pVal`	Pixel co-ordinate of the left window border

2.3.3.7.1 This property returns the position of the left window border.

2.3.3.8 Property *Top*

```
[propget, id(8)] HRESULT Top([out, retval] long *pVal)
```

Argument Name	Description
`pVal`	Pixel co-ordinate of the upper window border

2.3.3.8.1 This property returns the position of the upper window border.

2.3.3.9 Property *Right*

```
[propget, id(9)] HRESULT Right([out, retval] long *pVal)
```

Argument Name	Description
`pVal`	Pixel co-ordinate of the right window border

2.3.3.9.1 This property returns the position of the right window border.

2.3.3.10 Property *Bottom*

```
[propget, id(10)] HRESULT Bottom([out, retval] long *pVal)
```

Argument Name	Description
`pVal`	Pixel co-ordinate of lower window border

2.3.3.10.1 This property returns the position of the lower window border.

2.3.3.11 Property *VisualUpdateFrequency*

```
[propget, id(11)] HRESULT VisualUpdateFrequency([out, retval]
            float *pVal)

[propput, id(11)] HRESULT VisualUpdateFrequency([in] float
            newVal)
```

Argument Name	Description
`pVal, newVal`	Visual update frequency in Hz.

2.3.3.11.1 This property sets the visual update frequency for the view and retrieves its current setting. If `newVal` exceeds the maximum value of 10 Hz, this maximum value will be set and an error message will be generated (cf. 2.6.3.3).

2.3.3.12 Method *SaveScreenshot*

```
[id(12)] HRESULT SaveScreenshot([in] BSTR filename, [in] BOOL
            Overwrite)
```

Argument Name	Description
`Filename`	Absolute path name of the file in which the screenshot is to be stored as a bitmap. The file extension should be “bmp” but will not be corrected otherwise.
`Overwrite`	If the file already exists and `Overwrite` is TRUE, the file will be overwritten. If the file already exists and `Overwrite` is FALSE, the screenshot will not be saved .

2.3.3.12.1 This method captures a screenshot of the view and saves it in a bitmap file. If the file already exists, it will only be overwritten if `Overwrite` is TRUE.

2.3.3.12.2 Error: If the file already exists and `Overwrite` is FALSE, an error message will be generated (cf. 2.6.3.3).

2.3.3.13 Method *SaveAllDiagrams*

```
[id(13)] HRESULT SaveAllDiagrams([in] BSTR filename, [in] BOOL
            Overwrite)
```

Argument Name	Description
`filename`	Initial part of a file name, including the path but excluding the extension. Each diagram of the view is saved in Enhanced Metafile Format into a single file. The names of these files start with the given string which are extended by the RTD Driver to indicate the individual diagrams.
`Overwrite`	If any file already exists and `Overwrite` is TRUE, this file will be overwritten. If any file already exists and `Overwrite` is FALSE, the corresponding diagram will not be saved .

2.3.3.13.1 This method saves all diagrams displayed by appropriate Real-time Display ActiveX Controls into files.

2.3.3.13.2 Any already existing file will only be overwritten if `Overwrite` is TRUE.

2.3.3.13.3 The controls will save their diagrams according to the current value of the property IApplication::BlackAndWhite.

2.3.3.13.4 Error: If any file already exists and `Overwrite` is FALSE an error message will be generated (cf. 2.6.3.3).

2.3.3.14 Method *SaveDiagram*

```
[id(14)] HRESULT SaveDiagram([in] BSTR objectname, [in] BSTR
            filename, [in] BOOL Overwrite)
```

Argument Name	**Description**
`Objectname`	Case-insensitive DV object name of an appropriate Real-time Display ActiveX Control displaying a diagram. Note that the user may assign a name to each DV object of a display page using the RTD Editor.
`Filename`	Absolute path name of the file in which the diagram is to be saved in Enhanced Metafile Format. The file extension should be “emf” but will not be corrected otherwise.
`Overwrite`	If the file already exists and `Overwrite` is TRUE, the file will be overwritten. If the file already exists and `Overwrite` is FALSE, the diagram will not be saved .

2.3.3.14.1 This method saves a single diagram into an Enhanced Metafile Format file.

2.3.3.14.2 Any already existing file will only be overwritten if `Overwrite` is TRUE.

2.3.3.14.3 The control will save its diagram according to the current value of the property IApplication::BlackAndWhite.

2.3.3.14.4 Error: If the file already exists and `Overwrite` is FALSE, an error message will be generated (cf. 2.6.3.3).

2.3.3.14.5 Error: If there is more than one object with the given name, an arbitrary object with the given name will be processed, an error message will be generated (cf. 2.6.3.3).

2.3.3.14.6 Error: If the object with the given name is not a Real-time Display ActiveX Control, the method will have no effect, an error message will be generated (cf. 2.6.3.3).

2.3.3.15 Method *PrintScreenshot*

```
[id(15)] HRESULT PrintScreenshot()
```

This method has no arguments.

2.3.3.15.1 This method captures a screenshot and prints it using the Print Server.

2.3.3.16 Method *PrintAllDiagrams*

```
[id(16)] HRESULT PrintAllDiagrams ()
```

This method has no arguments.

2.3.3.16.1 This method prints all diagrams displayed by appropriate Real-time Display ActiveX Controls using the Print Server.

2.3.3.16.2 The controls will handle the printer as a black and white printer according to the current value of the property IApplication::BlackAndWhite.

2.3.3.17 Method *PrintDiagram*

```
[id(17)] HRESULT PrintDiagram([in] BSTR objectname)
```

Argument Name	**Description**
`objectname`	Case-insensitive DV object name of an appropriate (i.e. graphical) Real-time Display ActiveX Control displaying a diagram. Note that the user may assign a name to each DV object of a display page using the RTD Editor.

2.3.3.17.1 This method prints a single diagram using the Print Server.

2.3.3.17.2 The control will handle the printer as a black and white printer according to the current value of the property IApplication::BlackAndWhite.

2.3.3.17.3 Error: If there is more than one object with the given name, an arbitrary object with the given name will be processed, an error message will be generated (cf. 2.6.3.3).

2.3.3.17.4 Error: If the object with the given name is not a Real-time Display ActiveX Control, the method will have no effect and an error message will be generated (cf. 2.6.3.3).

2.3.3.18 Method *Pause*

```
[id(18)] HRESULT Pause()
```

This method has no arguments.

2.3.3.18.1 This method pauses data display for this view. If the view already is in pause state, this method has no effect.

2.3.3.19 Method *Resume*

```
[id(19)] HRESULT Resume()
```

This method has no arguments.

2.3.3.19.1 This method resumes data display for this view. If the view is not in pause state, this method has no effect.

2.3.3.20 Method *CreateFullsets*

```
[id(20)] HRESULT CreateFullsets([out, retval] IFullsets**
            pFullsets)
```

Argument Name	**Description**
`pFullsets`	points to an object implementing the interface *IFullsets* (cf. 2.5)

2.3.3.20.1 This method creates an empty object implementing the interface *IFullsets* (short: an *IFullsets* object) which can subsequently be filled with fullsets for display by the appropriate Real-time Display ActiveX Controls.

2.3.3.21 Method *ClearAllFullsets*

```
[id(21)] HRESULT ClearAllFullsets()
```

This method has no arguments.

2.3.3.21.1 This method removes all fullsets from all diagrams of this view.

2.3.3.21.2 Note that the *IFullsets* objects will neither be modified nor removed by this method.

2.3.3.21.3 Following any subsequent modification of the *IFullsets* objects these may be displayed again using their method *IFullsets::Display*.

2.3.4 Events Fired

2.3.4.1 This interface has no events.

2.3.5 Usage Conditions and Restrictions

2.3.5.1 When the method *Close* has been called, the object must not be used any more.

2.3.5.2 An example of a valid calling sequence is:

- Name
- put PageSequenceName – get PageSequenceName
- put PageName – get PageName
- SetPosition
- Pause
- PrintAllDiagrams – PrintDiagram – PrintScreenshot – SaveAllDiagrams – SaveDiagram - SaveScreenshot
- Resume
- get Left – get Top – get Right – get Bottom
- Reload
- CreateFullsets – CreateFullsets – CreateFullsets
- ClearAllFullsets
- Close

2.3.5.3 A view window will not be closed automatically when the associated interface has been released but on explicit request only using the method *Close*.

2.3.6 Persistent Data

2.3.6.1 Which of the data associated to the interface *IView* are persistently stored in the initialization file of the RTD Driver is described in the Engineering Specification of Real-time Display Driver.

2.3.7 EXAMPLE

2.3.7.1 The following example illustrates the usage of the interface *IView*:

```
View1.PageSequenceName = "EnduranceTest"
View1.PageName = "StartUp"
View2.PageName = "Oil pressure"
page3 = View3.PageName

View3.SetPosition 100, -1, 900, -1
     ' horizontal position remains unchanged
top = View3.Top   ' = 100

View1.Pause
```

```
View1.PrintAllDiagrams
View2.PrintDiagram "YXPlot1"
View1.SaveDiagram  "Chart3", "temperatures.emf"

Set fullsetGroup1 = view1.CreateFullsets
Set fullsetGroup2 = view1.CreateFullsets

view1.resume    ‘ case-insensitive!

view1.ClearAllFullsets

view1.Close
```

2.4 Interface *ISpotlightSticky*

2.4.1 General

2.4.1.1 The interface *ISpotlightSticky* represents a single spotlight sticky display window and helps creat and controll this window.

2.4.1.2 An object for this interface cannot be created directly but through the method IApplication::CreateSpotlightSticky only.

2.4.2 Design

2.4.2.1 This is a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.4.3 Methods and Properties

2.4.3.1 Method *AddChannel*

```
[id(1)] HRESULT AddChannel([in] BSTR channel)
```

Argument Name	**Description**
`channel`	Case-sensitive name of a channel

2.4.3.1.1 Error: If the maximum number (10) of channels has been reached prior to this method call, no channel will be added, an error message will be generated (cf. 2.6.3.3).

2.4.3.1.2 Error: If method *Display* has been called prior to this method call, no channel will be added, an error message will be generated (cf. 2.6.3.3).

2.4.3.2 Method *Display*

```
[id(2)] HRESULT Display()
```

This method has no arguments.

2.4.3.2.1 This method displays the spotlight sticky display window and starts cyclical data retrieval and display and returns immediately.

2.4.3.2.2 Following this method call, no further channels can be added.

2.4.3.3 Method *Close*

```
[id(3)] HRESULT Close()
```

This method has no arguments.

2.4.3.3.1 This method closes the spotlight sticky display window and deletes the associated object. Consequently, the pointer to the spotlight sticky display window becomes invalid and must not be used thereafter.

2.4.4 Events Fired

2.4.4.1 This interface has no events.

2.4.5 Usage Conditions and Restrictions

2.4.5.1 The method *AddChannel* must not be called more often than the maximum number of channels (10) that a spotlight sticky display window can display.

2.4.5.2 Method *AddChannel* must not be called following *Display*.

2.4.5.3 After method *Close* has been called, the object must not be used any more.

2.4.5.4 An example of a valid calling sequence is:

- AddChannel – AddChannel - AddChannel
- Display
- Close

2.4.5.5 A spotlight sticky display window will not be closed automatically when the associated interface has been released but on explicit request only using the method *Close*.

2.4.5.6 Therefore, an automation client should keep its connection to any spotlight sticky display window that he has created in order to be able to close it again since there is no other way of accessing it.

2.4.6 Persistent Data

2.4.6.1 There are no persistent data associated to this interface; in particular, the channel list of the spotlight sticky display window is not stored persistently.

2.4.7 EXAMPLE

2.4.7.1 The following example illustrates the usage of the interface *ISpotlightSticky* (let `SLS` be an object implementing the *ISpotlightSticky* interface):

```
SLS.AddChannel "T1ABC"
SLS.AddChannel "T2XYZ"
SLS.AddChannel "P5"

SLS.Display‘ display SLS with 3 channels

‘ do a lot more and wait some time
SLS.Close
```

2.5 Interface *IFullsets*

2.5.1 General

2.5.1.1 The interface *IFullsets* represents a group (set) of fullsets to be displayed by the appropriate Real-time Display ActiveX Controls of a single view. The interface helps specify the individual fullsets and some display attributes of the complete group as well as display the group.

2.5.1.2 An object for this interface cannot be created directly but through the method *IView::CreateFullsets* only.

2.5.2 Design

2.5.2.1 This is a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.3 Methods and Properties

2.5.3.1 Property *Color*

```
[propget, id(1)] HRESULT Color([out, retval] OLE_COLOR *pVal)
```

```
[propput, id(1)] HRESULT Color([in] OLE_COLOR newVal)
```

Argument Name	Description
`pVal, newVal`	Colour of curves and symbols used to display the fullsets

2.5.3.1.1 This property sets and retrieves the common colour used to display all fullsets of this group.

2.5.3.1.2 The currently displayed colour will not be changed until the method *Display* is called.

2.5.3.1.3 This property refers to the parameter Color of the method *IFullsets::put_Values* of the Real-time Display ActiveX Controls.

2.5.3.2 Property *MarkerType*

```
[propget, id(2)] HRESULT MarkerType([out, retval] BSTR *pVal)
```

```
[propput, id(2)] HRESULT MarkerType([in] BSTR newVal)
```

Argument Name	Description
`pVal, newVal`	Marker type (symbol type) used to display the fullsets. Case-insensitive string from the following list: • Square, • Diamond, • Cross, • Plus, • Triangle, • Pyramid, • Asterisk, • Circle

2.5.3.2.1 This property sets and retrieves the common type of symbols used to display all fullsets of this group.

2.5.3.2.2 The currently displayed symbol type will not be changed until the method *Display* is called.

2.5.3.2.3 This property refers to the parameter MarkerType of the method *IFullsets::put_Values* of the Real-time Display ActiveX Controls (see Interface Control Document of Real-time Display ActiveX Controls).

2.5.3.3 Property *Legend*

```
[propget, id(3)] HRESULT Legend([out, retval] BSTR *pVal)
```

```
[propput, id(3)] HRESULT Legend([in] BSTR newVal)
```

Argument Name	Description
`pVal, newVal`	Legend text used to describe the fullsets

2.5.3.3.1 This property sets and retrieves the legend text used to describe all fullsets of this group. The legend will be displayed in the appropriate diagrams.

2.5.3.3.2 The currently displayed legend text will not be changed until the method *Display* is called.

2.5.3.3.3 This property refers to the parameter Legend of the method *IFullsets::put_Values* of the Real-time Display ActiveX Controls.

2.5.3.4 Method *AddFullset*

```
[id(4)] HRESULT AddFullset( [in] BSTR Test, [in] long Event);
```

Argument Name	Description
`Test`	Test name, case-sensitive
`Event`	Event ID

2.5.3.4.1 This method adds a single fullset to the group. The fullset is uniquely defined by its test name and event ID, the latter required to be a fullset ID.

2.5.3.4.2 Note that the fullset will not be displayed until the method *Display* is called.

2.5.3.4.3 Error: If the specified fullset cannot be found in the database, the group of fullsets will not be altered, an error message will be generated (cf. 2.6.3.3).

2.5.3.5 Method *Display*

```
[id(5)] HRESULT Display()
```

This method has no arguments.

2.5.3.5.1 This method actually displays the group of fullsets.

2.5.4 Events Fired

2.5.4.1 This interface has no events.

2.5.5 Usage Conditions and Restrictions

2.5.5.1 The sequence of method calls and property calls is not restricted in any way. However, adding fullsets or changing display attributes will not become visually apparent until the method *Display* is called.

2.5.5.2 Displaying several groups of fullsets, each group to be displayed in a different colour or with different symbols, would be accomplished by creating several *IFullsets* objects.

2.5.5.3 An example of a valid calling sequence is:

- PutColor – PutMarkerType – PutLegend
- AddFullset - AddFullset - AddFullset
- Display

2.5.6 Persistent Data

2.5.6.1 There are no persistent data associated to this interface; in particular, the specification of the fullsets of this group is not stored persistently.

2.5.7 EXAMPLE

2.5.7.1 The following example illustrates the usage of the interface *IFullsets*:

```
fullsetGroup1.Color = vbRed
fullsetGroup1.MarkerType = "triangle"
fullsetGroup1.Legend = "test 1314"

fullsetGroup1.AddFullset ”test name 1”, 3333
fullsetGroup1.AddFullset ”test name 1”, 3336
fullsetGroup1.AddFullset ”test name 2”, 3339

fullsetGroup1.Display
```

2.6 Interface *IErrors*

2.6.1 General

2.6.1.1 The interface *IErrors* gives access to all error messages of the current connection to the RTD Driver. Each *IApplication* object has its own *IErrors* object. It is organised and accessed as a (FIFO) queue. Reading a message removes it from the queue.

2.6.1.2 If any method or property of the other interfaces described in this document fails for one of the documented reasons or any other business-logic reason, an immediately following call of *IErrors::LastCallHasFailed* will return TRUE. In addition, the failing methods generate an error message and add this to the error queue. In this case, the client should access the *IErrors* object and retrieve the individual error messages.

2.6.1.3 The methods and properties of this interface IErrors do neither set nor clear the error status flag, i.e. calling *IErrors::LastCallHasFailed* following any of the *IErrors* methods will return the error status of the most recent call to any method of the other interfaces.

2.6.1.4 An object for this interface cannot be created directly but its single object can be accessed through the (get) property *IApplication::Errors* only.

2.6.1.5 Note that any methods failing for technical reasons will throw appropriate exceptions, e.g. of type `_com_error`.

2.6.2 Design

2.6.2.1 This is a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.6.3 Methods and Properties

2.6.3.1 Method *LastCallHasFailed*

```
[id(1)] HRESULT LastCallHasFailed ([out, retval] BOOL * pbFailed)
```

Argument Name	**Description**
`pbFailed`	TRUE if the most recent call of any method or property of any of the other interfaces failed, otherwise FALSE.

2.6.3.1.1 This method should be called immediately following any other method or property call if the error status of this call is required, since any subsequent call will overwrite the error status.

2.6.3.2 Method *IsEmpty*

```
[id(2)] HRESULT IsEmpty([out, retval] BOOL *pbEmpty)
```

Argument Name	Description
`pbEmpty`	TRUE if currently there are no error messages, otherwise FALSE.

2.6.3.2.1 If this method returns TRUE, there are no current error messages.

2.6.3.3 Property *Message*

```
[propget, id(3)] HRESULT Message([out, retval] BSTR *pVal)
```

Argument Name	Description
`pVal`	The oldest error message in the queue.

2.6.3.3.1 This property retrieves the oldest error message in the queue and removes it from the queue.

2.6.3.3.2 If there is no error message, an empty string will be returned. In this case, no error message will be added to the queue. This erroneous access can be avoided by checking method *IsEmpty* prior to trying to retrieve the message text.

2.6.3.3.3 All error messages will be generated in the current language of the RTD Driver.

2.6.3.4 Method *Clear*

```
[id(4)] HRESULT Clear()
```

This method has no arguments.

2.6.3.4.1 This method removes all error messages although they may not have been retrieved yet.

2.6.4 Events Fired

2.6.4.1 This interface has no events.

2.6.5 Usage Conditions and Restrictions

2.6.5.1 The methods and properties of this interface will never generate an error message.

2.6.5.2 Trying to retrieve an error message from an empty queue will result in returning an empty string.

2.6.5.3 An example of a valid calling sequence is:

- LastCallHasFailed
- IsEmpty - Message – IsEmpty- Message - IsEmpty – Message
- Clear

2.6.6 ***Persistent Data***

2.6.6.1 There are no persistent data associated to this interface.

2.6.7 ***EXAMPLE***

2.6.7.1 The following example illustrates the usage of the interface *IErrors*:

```
Errors.Clear     ‘ clear all errors

‘ do something that generates new errors

if ( Errors.LastCallHasFailed ) then
   msgbox "Last call failed, error messages will follow:"
end if

‘ retrieve and display all new error messages:
while not Errors.IsEmpty
   msgbox Errors.Message
wend
```