Proprietary Notice

Revision History

Rev.	Date	Author	Sections Affected	Description
3	Mar 6, 2009	Th. Speer	Rev. History	A revision history table was added to this table.
3	Mar 6, 2009	Th. Speer	2.5.8	Additon of the Multi-Switch Control

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

1.1.1 ProDAS (Professional Data Acquisition System) is a data acquisition system for gas turbine test cells. This decument defines the technical requirements for the interfaces offered by the RTD ActiveX Controls.

1.1.2 The RTD ActiveX Controls are display objects used as part of the RTD System of proDAS. This RTD System also comprises the RTDD and RTDE (cf. Engineering Specification for Real-time Display Driver and Engineering Specification for Real-time Display Editor, respectively)

1.1.3 This document defines the COM interface to be used by the clients of the RTD ActiveX Controls.

1.2 Scope

1.2.1 This document is intended for programmers of the COM client components using the COM interface(s) specified herein as well as the programmers of the server program offering the COM interface(s).

1.3 Applicable Documents

Number	Title
ES78001.2620	Functional Requirements Document for ProDAS
ES78022.2766	Engineering Specification for Real-time Display ActiveX Controls
ES78021.2650	Engineering Specification for Real-time Display Editor
ES78022.2651	Engineering Specification for Real-time Display Driver

1.4 Abbreviations and Definitions

Term	Definition
CLSID	Class identification number
COM	Component Object Model
DISPID	Dispatch identification number
DLL	Dynamic Link Library
EMF	Enhanced Windows Metafile
ES	Engineering Specification
Fullset	Record of averaged data consisting of all channel values measured at one moment including all channel values calculated from those; we distinguish between historical and online fullsets
Graphical Real-time Display ActiveX Controls	Real-time Display ActiveX Controls containing diagrams
Hi	Upper yellow limit
HiHi	Upper red limit
Historical fullset	Fullset retrieved from the Test Result Database
ICD	Interface Control Document
IDL	Interface Definition Language
Lo	Lower yellow limit
LoLo	Lower red limit
N/A	Not applicable
online channel	Channel of which data values are cyclically delivered to a RTD ActiveX Control
online fullset	A fullset delivered by the Real-time Engine.

proDAS	Professional Data Acquisition System
ProgID	A string identifier which represents the class identifier (CLSID) of a COM/ActiveX object.
RTD	Real-time Display
RTD ActiveX Control	An ActiveX control specifically designed for use within the real-time display pages of proDAS
RTDD	Real-time Display Driver
RTDE	Real-time Display Editor
RTD Variable	A variable of a RTD ActiveX Control which is made public by its DISPID. All properties and methods of such a RTD Variable can be accessed via the general interface *IRTDVar* (cf. 2.3.2)
RTE	Real-time Engine
Real-time Display System	The partial software system within proDAS responsible for data visualisation
Sphinx Open	Name of the graphics library used for the graphical RTD ActiveX Controls

2. DESIGN

2.1 Introduction

2.1.1.1 Each RTD ActiveX Control will be made available as a separate DLL file.

2.1.1.2 The RTD ActiveX Controls are server components which shall be accessible via COM on the computer where they are registered.

2.1.1.3 The RTD ActiveX Controls shall be in-process servers.

2.1.1.4 It shall be possible to have an unlimited number (only depending on the available resources) of instances of every RTD ActiveX Control running simultaneously on a certain computer.

2.1.1.5 Note that a computer on which the RTD ActiveX Controls are used does not necessarily have to be part of a proDAS environment. The RTD ActiveX Controls can be used on any computer provided that they are registered there.

2.1.1.6 However, this document will only describe those interfaces of the various RTD ActiveX Controls that are relevant to the RTD clients of proDAS, namely the RTDE and RTDD.

2.1.1.7 Furthermore, standard interfaces that are inherited by the default interface class of each RTD ActiveX Control are not outlined in this document. Such a standard interface is *IPersistStreamInitImpl* which is used by all RTD ActiveX Controls to load and save persistent properties from or to a data stream, respectively.

2.2 Specific vs. general interfaces

2.2.1.1 Each RTD ActiveX Control has its own specific interface so that certain RTD Variables of the control are made public through their pre-defined DISPIDs as variables of type float or string.

2.2.1.2 This specific interface, which is the default interface of the control's ATL COM project, enables an interactive user to map his variables (i.e. channels) to those RTD Variables using a suitable mapping dialogue.

2.2.1.3 In addition to such a specific default interface, all RTD ActiveX Controls shall provide certain general interfaces which are used by the RTD clients, i.e. the RTDE and RTDD, for the communication with them.

2.2.1.4 This concept has the advantage that the RTD client does not have to know with which specific type of control it actually interacts.

2.2.1.5 Figure 1 gives an overview of these general interfaces provided by the RTD ActiveX Controls.

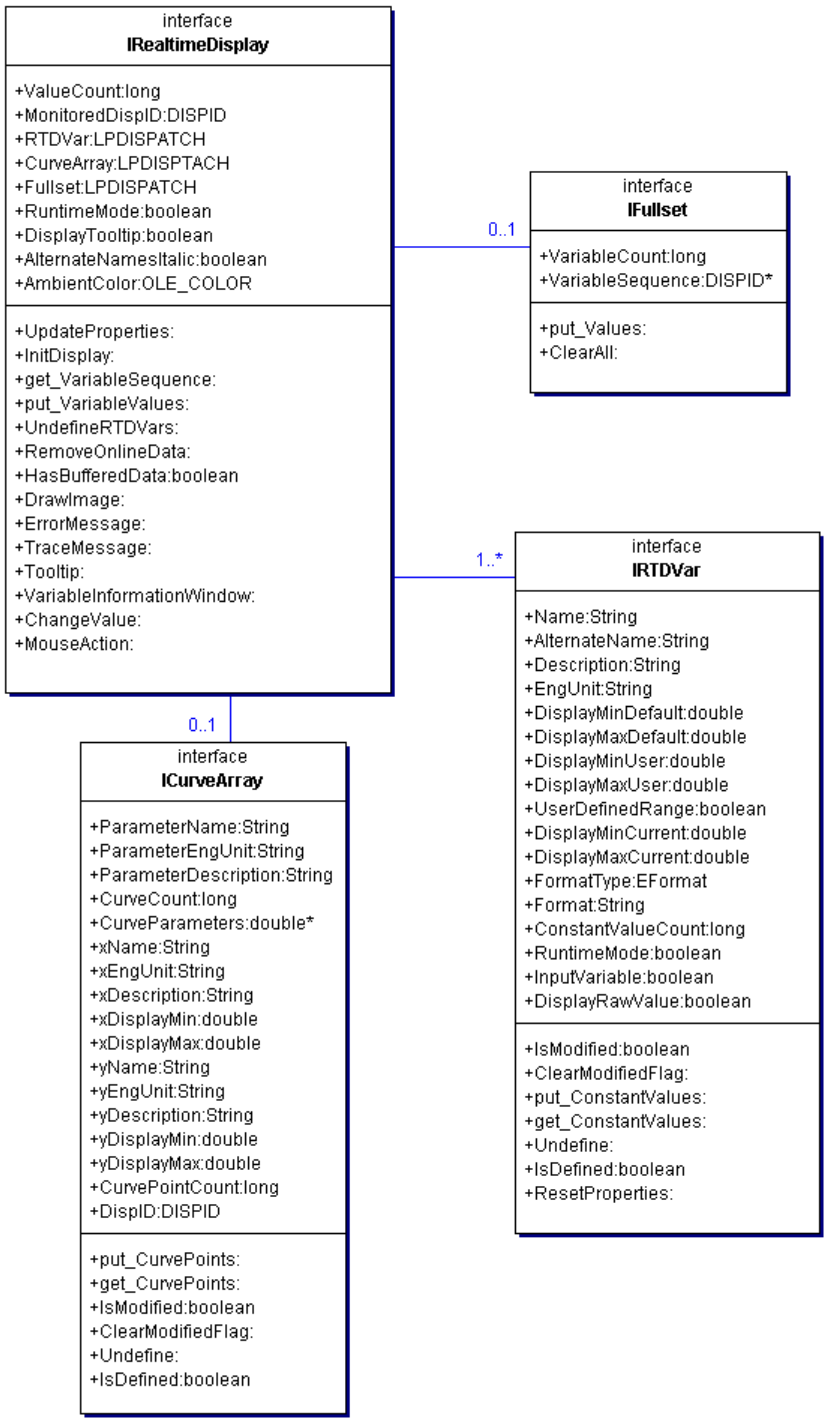

Figure 1: Overview of the general interfaces offered by the RTD ActiveX Controls

2.2.1.6 The RTD client identifies an arbitrary control as a RTD ActiveX Control, if it determines the general interface *IRealtimeDisplay* as being part of the latter.

2.2.1.7 After the RTD ActiveX Control has been identified as such, the RTD client uses this interface as a starting point for all further communication with the RTD ActiveX Control, i.e. the RTD client will use this interface to access other general interfaces that are also part of the control, as well as retrieve/set the properties and call the methods defined therein.

2.3 General interfaces

Whereas the interfaces specific to the individual RTD ActiveX Controls are outlined in section 2.5 of this document, the following subsections describe the general interfaces provided with the RTD ActiveX Controls in more detail.

2.3.1 Interface *IRealtimeDisplay*

2.3.1.1 Introduction

2.3.1.1.1 The interface *IRealtimeDisplay* implemented in all RTD ActiveX Controls is the starting point for a RTD client, i.e. the RTDE or RTDD, in the communication with such a control.

2.3.1.1.2 First of all, this interface is used to gain access to the RTD Variables of a RTD ActiveX Control. The number of such RTD Variables which a RTD ActiveX Control comprises depends on its individual type.

2.3.1.1.3 The properties and methods of each of these variables are offered by the interface *IRTDVar* (cf. 2.3.2). Such a single RTDVar object may be obtained from the property *RTDVar* (cf. 2.3.1.3.6) of the interface *IRealtimeDisplay* and modified by the client afterwards, unless the control is set to runtime mode (cf. 2.3.1.3.9).

2.3.1.1.4 Furthermore, the interface *IRealtimeDisplay* is used by the RTD clients to obtain information on whether or not certain features, such as the display of fullsets or three-dimensional breakpoint tables, are offered by the individual RTD ActiveX Control.

2.3.1.1.5 Finally, this interface is used by a RTD client for data transfer to the corresponding RTD ActiveX Control.

2.3.1.1.6 Note that although this interface is exported by all RTD ActiveX Controls, its methods and properties are implemented in each type of RTD ActiveX Control separately in order to accommodate their individual requirements and behaviour.

2.3.1.2 Design

2.3.1.2.1 This interface will be a dispatch interface.

2.3.1.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.3.1.2.3 The ProgID for this interface is *"RealtimeDisplayCtl.RealtimeDisplay"*.

2.3.1.3 Methods and Properties

2.3.1.3.1 Type definitions in IDL

2.3.1.3.1.1 The sequence in the following enumeration type `ELimit` must not be changed since it reflects the drawing sequence of the limit curves which are flooded by the corresponding colours for the individual alarm states.

```
typedef enum
{
Lo          = 0,
LoLo        = 1,
Hi          = 2,
HiHi        = 3,
Prim        = 4,
LimitCount  = 5   // LimitCount gives number of entries
} ELimit;
```

2.3.1.3.1.2 The enumeration type representing the different qualities for the value of a variable:

```
typedef enum
{
good        = 0,  // value is good
bad         = 1,  // value is bad
suspect     = 2,  // value is suspect
repeated    = 3   // value is repeated
} EQuality;
```

2.3.1.3.1.3 The enumeration type representing the different alarm states for the value of a variable:

```
typedef enum
{
OK                = 0,  // alarm status is ok
LoRed             = 1,  // Lower red alarm applies
LoYellow          = 2,  // Lower yellow alarm applies
HiYellow          = 3,  // Upper yellow alarm applies
HiRed             = 4,  // Upper red alarm applies
BOOLEAN_ALARM     = 5   // Discrete value has changed
} EAlarm;
```

2.3.1.3.2 Method *UpdateProperties*

```
[id(1), helpstring("method UpdateProperties")] HRESULT
UpdateProperties();
```

2.3.1.3.2.1 Calling this method notifies the RTD ActiveX Control that it has to re-read the properties of its RTD Variables and update the display accordingly.

2.3.1.3.2.2 This method has to be called by the RTD client indicating that mapping of client variables to RTD Variables of the control has been completed or after loading the properties from a persistent data source.

2.3.1.3.3 Property *ValueCount*

```
[propget, id(2), helpstring("property ValueCount")] HRESULT
ValueCount([out, retval] long *pVal);
```

Argument Name	Description
*pVal	Return the number of variables designated to be cyclically provided with data values.

2.3.1.3.4 Method *get_VariableSequence*

```
[id(3), helpstring("method get_VariableSequence")] HRESULT
get_VariableSequence(
[in] long lVars,
[out, size_is(lVars)] DISPID *pDispIDs,
[out, size_is(lVars)] ELimit *pLimits);
```

Argument Name	Description
lVars	Number of variables of the sequence. This number must be equal to that of the property *ValueCount* requested by the client previously (cf. 2.3.1.3.3)
*pDispIDs	Return the array with the DISPIDs of the RTD Variables designated to be cyclically provided with data values.
*pLimits	Return the array with the enumeration types each of them indicating one of the four alarm states or the normal state (cf. 2.3.1.3.1.1) for the corresponding RTD Variable.

2.3.1.3.4.1 After the RTD client has retrieved the number of the control's variables designated to be cyclically provided with data values from the property *ValuesCount* (cf. 2.3.1.3.3), it has to determine from this method in which order the control expects these variables to be provided with values.

2.3.1.3.4.2 Before the RTD client can query this property, it has to provide two arrays of length `lVars` (i.e. *ValueCount*), which are filled by the control with the DISPIDs and the corresponding limit states, respectively, of the RTD Variables.

2.3.1.3.5 Property *MonitoredDispID*

```
[propget, id(4), helpstring("property MonitoredDispID")] HRESULT
MonitoredDispID([out, retval] DISPID *pVal);
```

Argument Name	Description
`*pVal`	DISPID of the variable to be monitored by up to four other client variables (i.e. limit channels).

2.3.1.3.5.1 In case that one of the RTD Variables associated with the RTD ActiveX Control has to be monitored by others, a query of this property returns the DISPID of this RTD Variable.

2.3.1.3.5.2 If no RTD Variable has been marked as being monitored by others, querying this property will return the value `DISPID_UNKNOWN`.

2.3.1.3.5.3 Note that not more than one RTD Variable per RTD ActiveX Control can be marked as being monitored by others.

2.3.1.3.6 Property *RTDVar*

```
[propget, id(5), helpstring("property RTDVar")] HRESULT RTDVar([in]
DISPID DispID, [in] ELimit eLimit, [out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
`DispID`	DISPID of the variable to be retrieved by the client.
`eLimit`	The enumeration type indicating one of the four alarm states plus normal state (cf. 2.3.1.3.1.1).
`*pVal`	Return the pointer to the interface *IRTDVar* providing access to the methods and properties of the RTD Variable.

2.3.1.3.6.1 The properties and methods of a single RTD Variable are provided via the interface *IRTDVar* (cf. 2.3.2).

2.3.1.3.6.2 Depending on the value of the enumeration variable `ELimit`, a query of this property with the input argument `lDisp = DISPID_VALUE` returns a pointer to the interface of either the RTD Variable with that DISPID (in case of `ELimit = Prim`) or one of four other RTD Variables that are reserved by the control for being associated by the RTD client to up to four monitoring channels.

2.3.1.3.7 Property *CurveArray*

```
[propget, id(6), helpstring("property CurveArray")] HRESULT
CurveArray([out, retval] LPDISPATCH *pCurveArray);
```

Argument Name	Description
`*pCurveArray`	Return the pointer to the interface *ICurveArray* (cf. 2.3.4) or `NULL`, if this interface is not implemented.

2.3.1.3.7.1 Note that only the RTD Y/X Plot ActiveX control has this interface implemented and, therefore, will return an address different from `NULL`.

2.3.1.3.7.2 The interface *ICurveArray* (cf. 2.3.4) provides access to the property of a RTD ActiveX Control which is designated to be mapped to a three-dimensional breakpoint table object.

2.3.1.3.8 Property *Fullset*

```
[propget, id(7), helpstring("property Fullset")] HRESULT Fullset([out,
retval] LPDISPATCH * pFullset);
```

Argument Name	Description
`*pFullset`	Return the pointer to the interface *IFullset* (cf. 2.3.3) or `NULL`, if this interface is not implemented.

Note that only the RTD Y/X and Profile Plot ActiveX Controls will have the interface *IFullset* implemented and, thus, will return a value for the pointer `pFullset` different from `NULL`.

2.3.1.3.9 Property *RuntimeMode*

```
[propput, id(8), helpstring("property RuntimeMode")] HRESULT
RuntimeMode([in] BOOL newVal);
```

Argument Name	Description
`newVal`	The flag indicating whether the RTD ActiveX Control has to operate in runtime (`TRUE`) or edit (`FALSE`) mode.

2.3.1.3.9.1 Note that if this property is set to `TRUE`, i.e. indicating runtime mode of the RTD ActiveX Control, the runtime flag (cf. 2.3.2.3.22) of each RTDVar object belonging to this control will automatically be set to `TRUE`. As a result, any of the client's attempts to modify a property of such an RTDVar object will be ignored.

2.3.1.3.9.2 In addition, a RTD ActiveX Control in runtime mode will allow changes of only a few properties within its property dialogue.

2.3.1.3.9.3 The default value set by the constructors of all RTD ActiveX Controls shall be `FALSE` indicating Edit Mode. This enables a client to change all properties offered by the control via its various interfaces.

2.3.1.3.10 Method *InitDisplay*

```
[id(9), helpstring("method InitDisplay")] HRESULT InitDisplay([in]
DATE DateTime, [in] double dMilliseconds);
```

Argument Name	Description
`DateTime`	Argument of type DATE specifying the date and time stamp of the first data values after (re)start of the runtime mode of the RTD ActiveX Control
`dMilliseconds`	The milliseconds of the time stamp belonging to the argument DateTime (see above).

This method is used to set the starting time for the display of a RTD Strip Chart ActiveX Control.

2.3.1.3.11 Method *put_VariableValues*

```
[id(10), helpstring("method put_VariableValues")] HRESULT
put_VariableValues(
[in] long lScanCount,
[in] long lDisplayValues,
[in, size_is(lScanCount)] DATE *DateTime,
[in, size_is(lScanCount)] double *dMilliseconds,
[in, size_is(lScanCount*lDisplayValues)] double *dValues,
[in, size_is(lScanCount*lDisplayValues)] EQuality *eQualities,
[in, size_is(lScanCount*lDisplayValues)] EAlarm *eAlarmStates);
```

Argument Name	Description
`lScanCount`	Number of scans being transferred by the client.
`lDisplayValues`	Number of values per scan being transferred by the client
`*DateTime`	Vector of size `lScanCount` containing the date and time stamps (type DATE) of the scans being transferred
`*dMilliseconds`	Vector of size `lScanCount` containing the milliseconds of the date and time stamps of the scans being transferred
`*dValues`	Vector of size `(lScanCount*lDisplayValues)` containing the data values of all variables being transferred
`*eQualities`	Vector of size `(lScanCount*lDisplayValues)` containing the qualities of the data values being transferred via the vector `*fValues`. See paragraph 2.3.1.3.1.2 for valid values for this enumeration variable.
`*eAlarmStates`	Vector of size `(lScanCount*lDisplayValues)` containing the alarm states corresponding to the data values being transferred via the vector `*fValues`. See paragraph 2.3.1.3.1.3 for valid values for this enumeration variable.

2.3.1.3.11.1 The sequence of the elements in the vectors `fValues`, `eQualities`, and `eAlarmStates` is such that the values of all elements belonging to the same scan are ordered consecutively. Within one scan, the value sequence retrieved from the method *get_VariableSequence* (cf. 2.3.1.3.4) has to be taken into account.

2.3.1.3.11.2 Note that it depends on the individual type of RTD ActiveX Control in which way it will react on the quality enumeration value set by the client. For further details see the Engineering Specification for Real-time Display ActiveX Controls.

2.3.1.3.12 Method *UndefineRTDVars*

```
[id(11), helpstring("method UndefineRTDVars")] HRESULT
UndefineRTDVars();
```

A call of this method will automatically call the method *Undefine()* (cf. 2.3.2.3.20) of all RTDVar objects and (if existing) the *CurveArray* object (cf. 2.3.1.3.7) belonging to this RTD ActiveX Control.
Since the RTD client (i.e. the RTDE) will only set the properties for RTD Variables that are mapped to client variables (i.e. channels), the method *UndefineRTDVars* has to be called by the RTD client (i.e. the RTDE) before the RTD client sets the new values for the properties of the mapped RTD Variables. Otherwise, the control would still

display the data of certain RTD Variables which the user might have disconnected (i.e. unmapped) from the client variables in the preceding mapping process.

2.3.1.3.13 Method *RemoveOnlineData*

```
[id(12), helpstring("method RemoveOnlineData")] HRESULT
RemoveOnlineData();
```

A call of this method will remove all the previously displayed values or data points from the RTD ActiveX Control that correspond to online data, i.e. online channels and online fullsets.

2.3.1.3.14 Method *HasBufferedData*

```
[id(13), helpstring("method HasBufferedData")] HRESULT
HasBufferedData([out, retval] BOOL *pVal);
```

Argument Name	Description
`*pVal`	Return flag indicating whether (`TRUE`) or not (`FALSE`) this control has the capability of buffering data.

2.3.1.3.14.1 Note that only the graphical RTD ActiveX Controls, i.e. the RTD Y/X Plot, Profile Plot, and Strip Chart ActiveX Controls, have the capability to buffer data values.

2.3.1.3.14.2 Calling this method is useful, e.g. if the RTDD has paused an entire display page containing several display objects. Then, only the RTD ActiveX Controls able to buffer data have to be provided with data values while the display page remains paused. By doing so, the performance of the RTDD can be further optimized.

2.3.1.3.15 Method *DrawImage*

```
[id(14), helpstring("method DrawImage")] HRESULT DrawImage(
[in] OLE_HANDLE hDC,
[in] BOOL bBlackAndWhite,
[in] long lLeft,
[in] long lTop,
[in] long lRight,
[in] long lBottom);
```

Argument Name	Description
`hDC`	Device context (delivered by the client) to which the graphical RTD ActiveX Control will draw its image.

bBlackAndWhite	Flag indicating whether (TRUE) or not (FALSE) the RTD ActiveX Control shall add additional symbols to coloured curve lines in order to provide a clear distinction of the latter if the image is written to a device generating a black & white printout.
lLeft	Pixel co-ordinate of the left boundary of the output window of the device context to which the graphical RTD ActiveX Control has to transform from its drawing window.
lTop	Pixel co-ordinate of the top boundary of the output window of the device context.
lRight	Pixel co-ordinate of the right boundary of the output window of the device context.
lBottom	Pixel co-ordinate of the bottom boundary of the output window of the device context.

Note that only a graphical RTD ActiveX Control will react on a call of this method.

2.3.1.3.16 Property *DisplayTooltip*

```
[propput, id(15), helpstring("property DisplayTooltip")] HRESULT
DisplayTooltip([in] BOOL newVal);
```

Argument Name	Description
newVal	The flag indicating whether (TRUE) or not (FALSE) the RTD ActiveX Control has to request the display of a tooltip by firing the event *Tooltip* (cf. 2.3.1.4.3) when the user moves with the mouse over the numerical output field of one of its RTD Variables. Each RTD ActiveX Control will initialise with the value of this property set to FALSE.

2.3.1.3.17 Property *AlternateNameItalic*

```
[propput, id(16), helpstring("property AlternateNameItalic")] HRESULT
AlternateNameItalic([in] BOOL newVal);
```

Argument Name	Description
`newVal`	The flag indicating whether (`TRUE`) or not (`FALSE`) the alternate name has to be displayed with italicised font. Note that it is selectable in the property dialogue of each RTD ActiveX Control whether or not the alternate name of a channel (cf. 2.3.2.3.3) will be displayed instead of its name. The default value for the property *AlternateNameItalic* will be `FALSE`. Note that the value of this property will not be stored persistently.

2.3.1.3.18 Property *AmbientColor*

```
[propput, id(17), helpstring("property AmbientColor")] HRESULT
AmbientColor([in] OLE_COLOR newVal);
```

Argument Name	Description
`newVal`	The ambient color specified by the client application. Each RTD ActiveX Control will initially set the value of this property to that of its background colour.

This property has to be called by the client application by the time when the RTD ActiveX Control is created and whenever the client application changes the background colour of the window that contains the control. Only then, the RTD ActiveX Control will be able to adopt its background colour to that ambient colour whenever an interactive user sets its background style to be transparent with the property dialogue.

2.3.1.4 Events Fired

2.3.1.4.1 Event *ErrorMessage*

```
[id(1), helpstring("method ErrorMessage")] HRESULT ErrorMessage([in]
BSTR Message);
```

Argument Name	Description
`Message`	The string containing the error message, which occurred inside the RTD ActiveX Control, is sent to the client. The error message will either be sent in English or German language depending on the respective setting of the thread locale, which is queried by each RTD ActiveX Control.

2.3.1.4.2 Event *TraceMessage*

```
[id(2), helpstring("method TraceMessage")] HRESULT TraceMessage([in]
BSTR Message);
```

Argument Name	Description
`Message`	The string containing the trace message is sent to the client. This feature is especially useful for obtaining debug information from the control. This trace message will either be sent in English or German language depending on the respective setting of the thread locale, which is queried by each RTD ActiveX Control.

2.3.1.4.3 Event *Tooltip*

```
[id(3), helpstring("method Tooltip")] HRESULT Tooltip(
[in] BSTR VariableName,
[in] long lLeft,
[in] long lTop);
```

Argument Name	Description
`VariableName`	The name of the RTD Variable (i.e. the name of the channel associated with it) of which the RTD client shall display additional information in a tooltip.
`lLeft`	Pixel co-ordinate of the left boundary of the tooltip window that the client has to display.
`lTop`	Pixel co-ordinate of the top boundary of the tooltip window that the client has to display.

Whenever the user moves the mouse cursor over the numerical output field of the RTD Variable with the name `VariableName`, the RTD ActiveX Control will fire an event telling the client to display additional information about this variable in a tooltip at the specified position defined by the co-ordinates `lLeft` and `lTop` which represent the upper left corner of the tooltip window. Note that this event will only be fired, if the property *DisplayTooltip* (cf. 2.3.1.3.16) is set to the value `TRUE`.

2.3.1.4.4 Event *VariableInformationWindow*

```
[id(4), helpstring("method VariableInformationWindow")] HRESULT
VariableInformationWindow (
[in] long lLeft,
[in] long lTop);
```

Argument Name	Description
`lLeft`	Pixel co-ordinate of the left boundary of the variable information window that the client has to display.
`lTop`	Pixel co-ordinate of the top boundary of the variable information window that the client has to display.

When the user selects the display of the channel information window from the context menu of a RTD ActiveX Control, the latter will fire an event telling the client to display additional information about all of its RTD Variables in a window popping up at the specified position defined by the pixel co-ordinates `lLeft` and `lTop`.

2.3.1.4.5 Event *ChangeValue*

```
[id(5), helpstring("method ChangeValue")] HRESULT ChangeValue ([in]
BSTR VariableName, [in] double newVal);
```

Argument Name	Description
`VariableName`	The name of the client variable mapped to the RTD Variable of the control.
`newVal`	The new value set by an interactive user.

Note that this event is only fired by the RTD Input ActiveX Control when its property *Value* (cf. 2.5.7.3.1) has been changed by the interactive user.

2.3.1.4.6 Event *MouseAction*

```
[id(6), helpstring("method MouseAction")] HRESULT MouseAction(
[in] long uMsg,
[in] long wParam,
[in] long lParam);
```

Argument Name	Description
`uMsg`	The window message indicating which mouse action has been performed by the interactive user.
`wParam`	Parameter that indicates whether various virtual keys are down, i.e. whether or not the Shift and/or Ctrl keys were pressed during the mouse action.
`lParam`	Parameter that contains the x- (low-order word) and y- (high-order word) coordinates of the current position of the mouse pointer.

This event is used to notify the RTD client about a mouse action performed by the interactive user within the control's window. Whether or not the RTD ActiveX Control will actually fire this event depends not only on the value of the Runtime flag (cf. 2.3.1.3.9) but also on the specific mouse action.

2.3.1.4.7 Event *PrintControl*

```
[id(7), helpstring("method PrintControl")] HRESULT PrintControl();
```

This event is used to notify the RTD client that it should print the control.

2.3.1.4.8 Event *SaveControl*

```
[id(8), helpstring("method SaveControl")] HRESULT SaveControl();
```

This event is used to notify the RTD client that it should save the control into a file to be provided by the client.

2.3.1.4.9 Event *DisplayHelp*

```
[id(9), helpstring("method DisplayHelp")] HRESULT DisplayHelp([in]
BSTR HelpContext);
```

Argument Name	Description
`HelpContext`	Help context of the online documentation

This event is used to notify the RTD client that the online documentation should be displayed initially showing the topic associated with the help context given in the event. Note that this help context and the associated URL must be defined in the input file for the Online Help Server.

2.3.1.5 Usage Conditions and Restrictions

2.3.1.5.1 After the mapping process has been finished by the user and the RTD client has set the new properties of the RTD Variables, the method *UpdateProperties* (cf. 2.3.1.3.2) has to be called by the client first in order to notify the RTD ActiveX Control that its variables have changed. As a result, the latter will automatically update its display accordingly.

2.3.1.5.2 After the method *UpdateProperties* has been called, the RTD client is able to retrieve the number of RTD Variables mapped to online channels from the property *ValueCount* (cf. 2.3.1.3.3).

2.3.1.5.3 Since the method *get_VariableSequence* and the method *put_VariableValues* require the property value of *ValueCount* as input parameters, the latter must be called prior to those.

2.3.1.5.4 The pointer of type LPDISPATCH returned by the property *CurveArray* (cf. 2.3.1.3.7) ill only point to an existing object in case of the RTD Y/X Plot ActiveX Control. All other RTD ActiveX Controls will return `NULL`, since they are not able to process 3d breakpoint tables.

2.3.1.6 Persistent Data

There is no data stored persistently by the interface *IRealtimeDisplay*. Since the methods and properties of this interface are implemented separately in each RTD ActiveX Control (cf. 2.3.1.1.6), all the properties, which are provided through this interface and are relevant to the individual RTD ActiveX Control, are stored by the specific interface (cf. 2.5) of the latter.

2.3.1.7 Examples

2.3.1.7.1 Provide the RTD Variables of the control with some information about client variables (channels) mapped to them

The following example illustrates the usage of the interface *IRealtimeDisplay* by a C++ client, i.e. the RTDE, to deliver channel information to a RTD ActiveX Control after the interactive user has mapped some of the channels to a corresponding number of RTD Variables.

2.3.1.7.1.1 Check by the RTD client whether it deals with a RTD ActiveX Control:

```
IRealtimeDisplayPtr RealtimeDisplayPtr (GetControlLPDISPATCH());
if (RealtimeDisplayPtr == NULL)
   return; // not a RTD ActiveX Control
```

2.3.1.7.1.2 Tell the control the background colour of its container:

```
COLORREF AmbientColor = RGB (255,255,255);  // white background colour
RealtimeDisplayPtr -> PutAmbientColor (AmbientColor);
```

2.3.1.7.1.3 Set the defined flag of all RTD Variables of the control to FALSE. Note that calling this method is important after the mapping process has been finished, because otherwise the RTD ActiveX Control would still display the data of RTD Variables that have been disconnected from client variables and, thus, have their modification flag set to `TRUE` (cf. 2.3.1.3.12).

```
RealtimeDisplayPtr -> UndefineRTDVars();
```

2.3.1.7.1.4 First, use property *MonitoredDispID* (cf. 2.3.1.3.5) to determine whether one of the mapped RTD Variables shall be monitored. If so, the value of the DISPID retrieved will be greater than `0`:

```
// dispID and proDAS channel of monitored channel (maximum one
// channel may be monitored)
const DISPID dispIDMonitored = RealtimeDisplayPtr ->
                               GetMonitoredDispID();
IChannelPtr pMonitoredChannel = NULL;
```

2.3.1.7.1.5 After the RTDE has retrieved the DISPIDs of all the RTD Variables of the control which have been mapped to variables of either type float or string, it is able to access each of these RTDVar objects via the interface *IRTDVar* (cf. 2.3.2) by using the property *RTDVar* (cf. 2.3.1.3.6) of the interface *IRealtimeDisplay*:

```
IRTDVarPtr RTDVarPtr (RealtimeDisplayPtr ->
                      GetRTDVar(dispID, Prim));
```

2.3.1.7.1.6 Then provide the RTD Variable of the RTD ActiveX Control with some information about the channel mapped to it (for more details on the properties and methods defined in the interface *IRTDVar* see section 2.3.2):

```
RTDVarPtr->PutName         (pChannel->GetName());
RTDVarPtr->PutAlternateName(pChannel->GetAlternateName());
RTDVarPtr->PutEngUnit      (pChannel->GetEngineeringUnit());
RTDVarPtr->PutDisplayMin   (pChannel->GetDisplayMin());
RTDVarPtr->PutDisplayMax   (pChannel->GetDisplayMax());
```

2.3.1.7.1.7 Deliver information on the monitoring channels to the control:

```
if (dispID == dispIDMonitored)
{
   double dMonitoredDisplayMin, dMonitoredDisplayMax;
   dMonitoredDisplayMin = RTDVarPtr->GetDisplayMinCurrent();
   dMonitoredDisplayMax = RTDVarPtr->GetDisplayMaxCurrent();

   IAlarmsPtr pAlarms = pChannel->GetAlarms();
   for ( int i=1; i<=pAlarms->GetCount(); i++ )
   {
      IAlarmPtr pAlarm = pAlarms->GetItem(i);
      ELimit Limit = configuration.GetAlarmType (pAlarm);

      if (Limit < LimitCount )
      {
         CString sLimitChannel (static_cast<char*>
                                (pAlarm->GetLimitChannel()));
         IChannelPtr pLimit =
               configuration.GetIChannel(sLimitChannel);

         IRTDVarPtr pRTDVar (RealtimeDisplayPtr->
                             GetRTDVar(dispIDMonitored, Limit));

         pRTDVar->PutName(pLimit->GetName());
```

```
            pRTDVar->PutAlternateName(pLimit->GetAlternateName());
            pRTDVar->PutEngUnit(pLimit->GetEngineeringUnit());
            pRTDVar->PutDisplayMin(pLimit->GetDisplayMin());
            pRTDVar->PutValueMax(pLimit->GetDisplayMax());
         }
      }
   }
```

2.3.1.7.1.8 Finally, call the method *UpdateProperties* so that the RTD ActiveX Control re-reads the properties of its RTD Variables and displays all properties that have been modified and are set as defined.

```
RealtimeDisplayPtr -> UpdateProperties();
```

2.3.1.7.2 Transfer of data values to a RTD ActiveX Control

2.3.1.7.2.1 Notify the RTD ActiveX Control that runtime starts by setting the property *RuntimeMode* (cf. 2.3.1.3.9). Note that this indicates to the control that only a limited number of its properties may be modified during runtime.

```
RealtimeDisplayPtr -> PutRuntimeMode(TRUE);
```

2.3.1.7.2.2 Set the starting time for the display (only relevant to RTD Strip Chart ActiveX Controls) by calling the method *InitDisplay* (cf. 2.3.1.3.10):

```
RealtimeDisplayPtr->InitDisplay (COleDateTime::GetCurrentTime(),
                                 0.0f);
```

2.3.1.7.2.3 Retrieve the number and the sequence of the RTD Variables that have to be cyclically provided with data values:

```
long lValues = RealtimeDisplayPtr -> GetValueCount();
ASSERT (lValues > 0);

DISPID  *plValSeq    = new DISPID [lValues];
ELimit  *plValLimit  = new ELimit [lValues];
RealtimeDisplayPtr -> get_VariableSequence (lValues,
                                            plValSeq,
                                            plValLimit);
```

2.3.1.7.2.4 Transfer n scans to the control:

```
DATE     *pDateTime     = new DATE     [n];
double   *pdMilliseconds= new float    [n];
double   *pdValues      = new float    [n*lValues];
EQuality *peQualities   = new EQuality [n*lValues];
ELimit   *peAlarmStates = new ELimit   [n*lValues];
// Transfer all values at once to the RTD ActiveX Control
RealtimeDisplayPtr -> put_VariableValues (n,
                                          lValues,
                                          pDateTime,
                                          pdMilliseconds,
                                          pdValues,
```

```
                                                peQualities,
                                                peAlarmStates);
```

2.3.2 Interface *IRTDVar*

2.3.2.1 Introduction

2.3.2.1.1 This interface is used for access to the RTD variables of each RTD ActiveX Control.

2.3.2.1.2 All properties and methods provided by this interface have an implementation common to all RTD ActiveX Controls in contrast to the interface *IRealtimeDisplay* (see paragraph 2.3.1.1.6).

2.3.2.1.3 There will be one instance of this interface created for each RTD Variable of the RTD ActiveX Control.

2.3.2.2 Design

2.3.2.2.1 This interface will be a dispatch interface.

2.3.2.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.3.2.2.3 The ProgID for this interface is *"RTDVariablesCtl.RTDVar"*.

2.3.2.3 Methods and Properties

2.3.2.3.1 Type definition in IDL

```
// Valid format types
typedef enum
{
Float       = 0,
Boolean     = 1,
Integer     = 2,
Date        = 3,
Time        = 4
}
EFormat;
```

2.3.2.3.2 Property *Name*

```
[propget, id(1), helpstring("property Name")] HRESULT Name([out,
retval] BSTR *pVal);
[propput, id(1), helpstring("property Name")] HRESULT Name([in] BSTR
newVal);
```

Argument Name	Description

`*pVal, newVal`	The name of the RTDVar object

2.3.2.3.3 Property *AlternateName*

```
[propget, id(2), helpstring("property AlternateName")] HRESULT
AlternateName([out, retval] BSTR *pVal);
[propput, id(2), helpstring("property AlternateName")] HRESULT
AlternateName([in] BSTR newVal);
```

Argument Name	Description
`*pVal, newVal`	The alternate name of the RTDVar object

2.3.2.3.4 Property *Description*

```
[propget, id(3), helpstring("property Description")] HRESULT
Description([out, retval] BSTR *pVal);
[propput, id(3), helpstring("property Description")] HRESULT
Description([in] BSTR newVal);
```

Argument Name	Description
`*pVal, newVal`	The string containing the current description of the RTDVar object

2.3.2.3.5 Property *EngUnit*

```
[propget, id(4), helpstring("property EngUnit")] HRESULT EngUnit([out,
retval] BSTR *pVal);
[propput, id(4), helpstring("property EngUnit")] HRESULT EngUnit([in]
BSTR newVal);
```

Argument Name	Description
`*pVal, newVal`	The string containing the current engineering unit of the RTDVar object

2.3.2.3.6 Property *DisplayMinDefault*

```
[propget, id(5), helpstring("property DisplayMinDefault")] HRESULT
DisplayMinDefault([out, retval] double *pVal);
[propput, id(5), helpstring("property DisplayMinDefault")] HRESULT
DisplayMinDefault([in] double newVal);
```

Argument Name	Description

`*pVal, newVal`	The minimum display value (of type double) of the RTDVar object. This property will be initialised with the value `0.0`.

2.3.2.3.7 Property *DisplayMaxDefault*

```
[propget, id(6), helpstring("property DisplayMaxDefault")] HRESULT
DisplayMaxDefault([out, retval] double *pVal);
[propput, id(6), helpstring("property DisplayMaxDefault")] HRESULT
DisplayMaxDefault([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The maximum display value (of type double) of the RTDVar object. This property will be initialised with the value `1.0`.

The display range specified by the properties *DisplayMinDefault* (cf. 2.3.2.3.6) and *DisplayMaxDefault* (cf. 2.3.2.3.7) represents the default display range of the client variable mapped to the RTDVar object. Therefore, it has to be set by the client application after the mapping process has been completed.

2.3.2.3.8 Property *DisplayMinUser*

```
[propget, id(7), helpstring("property DisplayMinUser")] HRESULT
DisplayMinUser([out, retval] double *pVal);
[propput, id(7), helpstring("property DisplayMinUser")] HRESULT
DisplayMinUser([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The minimum display value (of type double) of the RTDVar object specified by the interactive user.

2.3.2.3.9 Property *DisplayMaxUser*

```
[propget, id(8), helpstring("property DisplayMaxUser")] HRESULT
DisplayMaxUser([out, retval] double *pVal);
[propput, id(8), helpstring("property DisplayMaxUser")] HRESULT
DisplayMaxUser([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The maximum display value (of type double) of the RTDVar object specified by the interactive user.

The display range specified by the properties *DisplayMinUser* (cf. 2.3.2.3.8) and *DisplayMaxUser* (cf. 2.3.2.3.9) represents the display range of the RTDVar object specified by the interactive user, e.g. in the property dialogue of the RTD ActiveX Control.

2.3.2.3.10 Property *UserDefinedRange*

```
[propget, id(9), helpstring("property UserDefinedRange")] HRESULT
UserDefinedRange([out, retval] BOOL *pVal);
[propput, id(9), helpstring("property UserDefinedRange")] HRESULT
UserDefinedRange([in] BOOL newVal);
```

Argument Name	Description
`*pVal, newVal`	The flag indicating whether (`TRUE`) the user defined range specified by the properties *DisplayMinUser* (cf. 2.3.2.3.8) and *DisplayMaxUser* (cf. 2.3.2.3.9) or (`FALSE`) the range which is defined by the RTD Client via the properties *DisplayMinDefault* (cf. 2.3.2.3.6) and *DisplayMaxDefault* (cf. 2.3.2.3.7) is applied for the data display of this RTDVar object. The default value for the property *UserDefinedRange* will be `FALSE`.

2.3.2.3.11 Property *DisplayMinCurrent*

```
[propget, id(10), helpstring("property DisplayMinCurrent")] HRESULT
DisplayMinCurrent([out, retval] double *pVal);
[propput, id(10), helpstring("property DisplayMinCurrent")] HRESULT
DisplayMinCurrent([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The current minimum display value of the RTDVar object.

2.3.2.3.12 Property *DisplayMaxCurrent*

```
[propget, id(11), helpstring("property DisplayMaxCurrent")] HRESULT
DisplayMaxCurrent([out, retval] double *pVal);
[propput, id(11), helpstring("property DisplayMaxCurrent")] HRESULT
DisplayMaxCurrent([in] double newVal);
```

Argument Name	Description
`*pVal`	The current maximum display value of the RTDVar object.

The current display range of the RTDVar object in the RTD ActiveX Control is specified by the properties *DisplayMinCurrent* (cf. 2.3.2.3.11) and *DisplayMaxCurrent* (cf. 2.3.2.3.12). Before the RTD ActiveX Control (i.e. the RTDVar object) is supplied with data values, this range will be equal to that defined by the properties *DisplayMinDefault* (cf. 2.3.2.3.6) and *DisplayMaxDefault* (cf. 2.3.2.3.7), if the property *UserDefinedRange* (cf. 2.3.2.3.10) is set to `FALSE`, or, if `TRUE`, to that defined by *DisplayMinUser* (cf. 2.3.2.3.8) and *DisplayMaxUser* (cf. 2.3.2.3.9). During runtime however, the current display range may differ from any of those initial display ranges, e.g. due to automatic range increase performed by the RTD ActiveX Control. Hereby, the control uses the put properties of *DisplayMinCurrent* and *DisplayMaxCurrent*, whereas the RTD client is able to retrieve their current settings via the respective get properties

2.3.2.3.13 Property *FormatType*

```
[propget, id(12), helpstring("property FormatType")] HRESULT
FormatType([out, retval] EFormat *pVal);
[propput, id(12), helpstring("property FormatType")] HRESULT
FormatType([in] EFormat newVal);
```

Argument Name	Description
`*pVal, newVal`	The enumeration value (cf. 2.3.2.3.1) for the format type of the RTDVar object.

2.3.2.3.14 Property *Format*

```
[propget, id(13), helpstring("property Format")] HRESULT Format([out,
retval] BSTR *pVal);
[propput, id(13), helpstring("property Format")] HRESULT Format([in]
BSTR newVal);
```

Argument Name	Description
`*pVal, newVal`	The format string for the display of the value of the RTDVar object.

In case of a variable of format type discrete (cf. 2.3.2.3.13), the format string consists of various format strings associated with the respective enumeration values. These strings have to be separated from each other by the delimiter "|".

2.3.2.3.15 Method *IsModified*

```
[id(14), helpstring("method IsModified")] HRESULT IsModified([out,
retval] BOOL *pVal);
```

Argument Name	Description
`*pVal`	Return flag indicating whether (`TRUE`) or not (`FALSE`) a property of the RTDVar object has been modified by the client. This flag is set to `FALSE` by a call of the method *ClearModifiedFlag* (cf. 2.3.2.3.16).

After the RTD client has called the method *UpdateProperties* (cf. 2.3.1.3.2) which is part of the interface *IRealtimeDisplay* (cf. 2.3.1), this method is mainly used by the RTD ActiveX Control to determine whether the corresponding RTD Variable object has been modified or not.

2.3.2.3.16 Method *ClearModifiedFlag*

```
[id(15), helpstring("method ClearModifiedFlag")] HRESULT
ClearModifiedFlag();
```

Whenever this method is called, the modified flag which can be retrieved via the property *IsModified* (cf. 2.3.2.3.15) is set to `FALSE`.

2.3.2.3.17 Method *put_ConstantValues*

```
[id(16), helpstring("method put_ConstantValues")] HRESULT
put_ConstantValues([in] long lCount, [in, size_is(lCount)] double
*dValues);
```

Argument Name	Description
`lCount`	The number of constant values that are to be stored in this RTDVar object. After being set, this number is retrievable from the property *ConstantValueCount* (cf. 2.3.2.3.18).
`*dValues`	The vector of size `lCount` containing the constant values that are to be stored in this RTDVar object.

When the corresponding RTD Variable is mapped to either the x or y variable of a polynomial or two-dimensional breakpoint table, this method is used by the RTD client to store the array of values for this variable. In addition, it is used for a fixed limit value.

2.3.2.3.18 Property *ConstantValueCount*

```
[propget, id(17), helpstring("property ConstantValueCount")] HRESULT
ConstantValueCount([out, retval] long *plCount);
```

Argument Name	Description
`*plCount`	Return the number of constant values that have been stored in this RTDVar object.

2.3.2.3.19 Method *get_ConstantValues*

```
[id(18), helpstring("method get_ConstantValues")] HRESULT
get_ConstantValues([in] long lCount, [out, size_is(lCount)] double
*ppfValues);
```

Argument Name	Description
`lCount`	The number of constant values that have been stored in this RTDVar object. This input parameter has to be retrieved from the property *ConstantValueCount* (cf. 2.3.2.3.18).
`*ppfValues`	Return a vector of size `lCount` containing the constant values that have been stored in this RTDVar object by the client.

Note that the client has to provide the array with the length `lCount`. This method will just set a value for every element.

2.3.2.3.20 Method *Undefine*

```
[id(19), helpstring("method Undefine")] HRESULT Undefine();
```

Whenever this method is called, the defined flag of the RTD Variable is set to `FALSE`. This indicates to the RTD ActiveX Control not to display this variable at all.

2.3.2.3.21 Method *IsDefined*

```
[id(20), helpstring("method IsDefined")] HRESULT IsDefined([out,
retval] BOOL *pVal);
```

Argument Name	Description
`*pVal`	Return flag indicating whether (`TRUE`) or not (`FALSE`) this RTDVar object is defined.

2.3.2.3.22 Property *RuntimeMode*

```
[propget, id(21), helpstring("property RuntimeMode")] HRESULT
RuntimeMode([out, retval] BOOL *pVal);
[propput, id(21), helpstring("property RuntimeMode")] HRESULT
RuntimeMode([in] BOOL newVal);
```

Argument Name	Description
*pVal, newVal	The flag indicating whether (TRUE) or not (FALSE) the properties of this RTD Variable may be changed.

2.3.2.3.22.1 If this flag is set to TRUE, any attempt of a client to modify a property of this RTDVar object will be ignored.

2.3.2.3.22.2 This flag will automatically be set to TRUE in all RTDVar objects belonging to a RTD ActiveX Control, if the client application changes the property *RuntimeMode* (cf. 2.3.1.3.9) via the interface *IRealtimeDisplay* to the value TRUE.

2.3.2.3.23 Property *InputVariable*

```
[propget, id(22), helpstring("property InputVariable")] HRESULT
InputVariable([out, retval] BOOL *pVal);
[propput, id(22), helpstring("property InputVariable")] HRESULT
InputVariable([in] BOOL newVal);
```

Argument Name	Description
*pVal, newVal	The flag indicating whether (TRUE) or not (FALSE) the client channel mapped to this RTD Variable may be changed by an interactive user.

This property is used to indicate or query, respectively, whether or not an input channel is mapped to this RTD Variable. The value of an input channel may be changed by an interactive user during runtime. The RTD client will be notified about any value modification of such an input channel by the RTD ActiveX Control firing the event *ChangeValue* (cf. 2.3.1.4.5) which is part of the interface *IRealtimeDisplay* (cf. 2.3.1).

2.3.2.3.24 Property *DisplayRawValue*

```
[propget, id(23), helpstring("property DisplayRawValue")] HRESULT
DisplayRawValue([out, retval] BOOL *pVal);
[propput, id(23), helpstring("property DisplayRawValue")] HRESULT
DisplayRawValue([in] BOOL newVal);
```

Argument Name	Description
*pVal, newVal	The flag indicating whether (TRUE) the raw or (FALSE) the engineering unit converted value of the RTDVar object will be displayed during runtime.

2.3.2.3.25 Method *ResetProperties*

```
[id(24), helpstring("method ResetProperties")] HRESULT
ResetProperties();
```

This method resets certain properties (cf. 2.3.2.6) of the RTDVar object to their persistent values. If no data has been stored persistently before, these values will be set to their defaults as defined in the constructor of the object class. Note that this method is designated to be only called by the control itself, whenever an interactive user requests a reset of the properties of all RTD Variables in the property dialogue of the RTD ActiveX Control

2.3.2.4 Events Fired

There are no events fired by this interface.

2.3.2.5 Usage Conditions and Restrictions

2.3.2.5.1 If the readonly flag of the RTDVar object is set to `TRUE` (cf. 2.3.2.3.22), any attempt of a client to modify a property via this interface will be ignored.

2.3.2.5.2 The readonly flags of all RTDVar objects part of a RTD ActiveX Control will automatically be set to `TRUE`, if the client application changes the property *RuntimeMode* (cf. 2.3.1.3.9) to the value `TRUE`, which is accessible via the interface *IRealtimeDisplay*.

2.3.2.6 Persistent Data

2.3.2.6.1 Only those properties of a RTDVar object are stored persistently which an interactive user is able to modify in the property dialogue of the RTD ActiveX Control. The other properties are explicitly set by the RTD client whenever the RTD ActiveX Control and its associated RTD Variables are created.

2.3.2.6.2 The properties stored persistently are the following:

- *UserDefinedRange* (cf. 2.3.2.3.10),
- *DisplayMinUser* (cf. 2.3.2.3.8),
- *DisplayMaxUser* (cf. 2.3.2.3.9),
- *Format* (cf. 2.3.2.3.14), and
- *DisplayRawValue* (cf. 2.3.2.3.24).

2.3.3 Interface *IFullset*

2.3.3.1 Introduction

2.3.3.1.1 The RTD client shall connect to this interface via the property *Fullset* (cf. 2.3.1.3.8) of the interface *IRealtimeDisplay*.

2.3.3.1.2 If the RTD client obtains an address different from `NULL` for the pointer to the object represented by *IFullset*, it knows that this RTD ActiveX Control supports the display of archived fullsets.

2.3.3.1.3 First of all, the interface *IFullset* is used by the RTD client to retrieve the information which of the online curves of a RTD Y/X or Profile Plot ActiveX Control shall be overlayed with others that are set up with the same x and y variables but with the corresponding data values originating from the Windows Database.

2.3.3.1.4 The RTD client uses this interface to transfer the values of the archived fullsets as well as the layout properties of the newly added curves to the control.

2.3.3.2 Design

2.3.3.2.1 This interface will be a dispatch interface.

2.3.3.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.3.3.2.3 The ProgID for this interface is *"FullsetCtl.Fullset"*.

2.3.3.2.4 This interface will be implemented separately in the RTD Y/X Plot and Profile Plot ActiveX Control in order to accommodate their different requirements and behaviours (cf. 2.3.3.5.2).

2.3.3.3 Methods and properties

2.3.3.3.1 Type definition in IDL

```
// Valid marker symbols
typedef enum
{
Square      = 0,
Diamond     = 1,
Cross       = 2,
Plus        = 3,
Triangle    = 4,
Pyramid     = 5,
Asterisk    = 6,
Circle      = 7
}
```

```
EMarkerType;
```

2.3.3.3.2 Property *VariableCount*

```
[propget, id(1), helpstring("property VariableCount")] HRESULT
VariableCount([out, retval] long *pVal);
```

Argument Name	Description
`*pVal`	Return the number of RTD variables mapped to online channels for which the RTDD shall retrieve values from the Windows Database Server.

2.3.3.3.3 Property *VariableSequence*

```
[propget, id(2), helpstring("property VariableSequence")] HRESULT
VariableSequence([in] long lVars, [size_is(lVars), out, retval] DISPID
*pVal);
```

Argument Name	Description
`lVars`	Number of variables for which the sequence has to be retrieved. The client is able to obtain the value of this input argument from the property *VariableCount* (cf. 2.3.3.3.2).
`*pVal`	Return the array of size `lVars` containing the DISPIDs of the variables mapped to online channels.

Note that the client has to provide the array `pVal` of size `lVars` after the latter has been derived from the property *VariableCount* (cf. 2.3.3.3.2). The RTD ActiveX Control will only fill this array with the corresponding values.

2.3.3.3.4 Method *put_Values*

```
[id(3), helpstring("method put_Values")] HRESULT put_Values(
[in] long lFullsets,
[in] long lVars,
[in, size_is(lFullsets*lVars)] double *dValues,
[in] OLE_COLOR Color,
[in] EMarkerType MarkerType,
[in] BSTR Legend);
```

Argument Name	Description
`lFullsets`	Number of fullsets belonging to this subset.
`lVars`	Number of values per fullset. The RTD client has obtained the value for this input argument from the property *VariableCount* (cf. 2.3.3.3.2) in a preceding step.
`dValues`	The array of size `lFullsets*lVars` containing the values of all variables for display.
`Color`	The colour of the curves and point markers of this subset of fullsets. This colour value applies to all curve lines (each of them displaying one fullset) and point marker symbols.
`MarkerType`	The value of the enumeration type `EMarkerType` (cf. 2.3.3.3.1) for the marker type for the data points of this subset of fullsets.
`Legend`	The legend string displayed in the diagram of the RTD Y/X Plot or Profile Plot ActiveX Control, respectively. This legend provides some description on this subset of fullsets.

2.3.3.3.4.1 The sequence of the elements in the vector `fValues` is such that the values of all elements belonging to the same fullset are ordered consecutively, thus describing the transformation of a two-dimensional array `fValues[lFullset][lVars]`. Within one fullset, the sequence of the values has to be according to that derived from the property *VariableSequence* (cf. 2.3.3.3.3).

2.3.3.3.4.2 Note that the value of the argument `MarkerType` will only be applied in case of the RTD Profile Plot ActiveX Control. In case of the RTD Y/X Plot ActiveX Control, the same point marker symbol will be applied as for the corresponding online data point. In the latter case, only the colour setting (argument `Color`) will be applied for distinction purposes.

2.3.3.3.5 Method *ClearAll*

```
[id(4), helpstring("method ClearAll")] HRESULT ClearAll();
```

A call of this method will remove all previously displayed archived fullsets from the RTD ActiveX Control.

2.3.3.4 Events Fired

There are no events fired by this interface.

2.3.3.5 Usage Conditions and Restrictions

2.3.3.5.1 There are no special restrictions in using this interface.

2.3.3.5.2 During runtime, the RTD client, i.e. the RTDD asks a RTD ActiveX Control for the interface *IFullset* via the property *Fullset* (cf. 2.3.1.3.8). It will only receive a valid address of this interface in case of either a RTD Y/X Plot or Profile Plot ActiveX Control, which both have this interface implemented.

2.3.3.5.3 When an interactive user selects a subset of fullsets for display, the RTD client shall retrieve the number of RTD variables from the control which shall be overlayed with fullset data from the Windows Database via the property *VariableCount* (cf. 2.3.3.3.2). The RTD Variables which are to be overlayed by fullset data during runtime have been selected by an interactive user with the property dialogue of the RTD ActiveX Control.

2.3.3.5.4 Note that if no RTD Variables of a RTD Y/X Plot or Profile Plot ActiveX Control have previously been marked in the property dialogue for overlay with fullset data, the RTD client will receive the value `0` for this property and no fullset data can be displayed.

2.3.3.5.5 In a subsequent step, a query of the property *VariableSequence* (cf. 2.3.3.3.3) provides the RTD client with an array of DISPIDs of those RTD Variables. The sequence of the array's elements specifies the order in which the RTD ActiveX Control expects the values for the variables.

2.3.3.5.6 With the DISPIDs retrieved from this property, the RTD client can obtain the objects of these RTD Variables from the property *RTDVar* (cf. 2.3.1.3.6) part of the interface *IRealtimeDisplay* and query all information necessary from them.

2.3.3.5.7 Finally, a call of the method *put_Values* (cf. 2.3.3.3.4) is used to display one subset of fullsets in the diagram of the control. Any subsequent call of this method will add the data points of further fullsets to the diagram until they are removed by a call of the method *ClearAll* (cf. 2.3.3.3.5) that is also part of the interface *IRealtimeDisplay*.

2.3.3.5.8 Note that hereby, the total number of curves (including fullset and online curves) cannot exceed the number of two hundred due to the limitation of the graphics library Sphinx Open used by the graphical RTD ActiveX Controls.

2.3.3.6 Persistent Data

There is no data stored persistently by this interface. Since the latter is implemented separately in the RTD Y/X and Profile Plot ActiveX Controls (cf. 2.3.3.2.4), the boolean flags indicating for all RTD Variables of a certain RTD ActiveX Control whether or not they are marked for overlay with fullset data are stored in the interfaces *IYXPlot* (cf. 2.5.4) and *IProfilePlot* (cf. 2.5.5), respectively.

2.3.3.7 Example

2.3.3.7.1 Access the interface *IFullset* of a RTD ActiveX Control and transfer the data values of one fullset

This example shows how a VC++ client, i.e. the RTDD, accesses the interface *IFullset* via a query of the property *Fullset* (cf. 2.3.1.3.8) which is part of the interface *IRealtimeDisplay* (cf. 2.3.1).

2.3.3.7.1.1 The RTD client checks first whether it deals with a RTD ActiveX control:

```
IRealtimeDisplayPtr RealtimeDisplayPtr (GetControlLPDISPTACH());
if (RealtimeDisplayPtr == NULL)
   return;    // not a RTD ActiveX Control
```

2.3.3.7.1.2 The RTD client queries the pointer to the interface *IFullset*. In case that it receives a valid address of this object it will use it to transfer fullset data:

```
IFullsetPtr FullsetPtr (RealtimeDisplayPtr->GetFullset());
if (FullsetPtr != NULL)
{
   // Check whether any RTD Variables of the control are marked
   // be overlayed with archived fullset data
   long lCount = FullsetPtr->GetVariableCount();
   if (lCount > 0)
   {
      DISPID *pDispIDs = new DISPID [lCount];
      FullsetPtr -> get_VariableSequence (lCount, pDispIDs);
      double *pValues = new double [lCount];
      for (int i = 0; i < lCount; i++)
      {
         IRTDVarPtr RTDVarPtr(
                  RealtimeDisplayPtr->GetRTDVar(pDispIDs[i],
                                                ELimit::Prim));
         pValues[i] = GetFullsetValue(RTDVarPtr->GetName());
      }

      // Transfer the values for one fullset to the Control
      FullsetPtr->put_Values(
                              1,
                              lCount,
                              pValues,
                              GetSymbolColor(),
                              GetMarkerType(),
                              GetLegend());
   }
}
```

2.3.4 Interface *ICurveArray*

2.3.4.1 Introduction

2.3.4.1.1 This interface is used to display an array of curves resulting from the projection of a three-dimensional breakpoint table to a two dimensional Y/X plot.

2.3.4.1.2 This interface is used to access the RTD ActiveX Control's property that is designated to be mapped to a three-dimensional breakpoint table.

2.3.4.1.3 This interface is only imported by the RTD Y/X Plot ActiveX Control.

2.3.4.1.4 Therefore, only this control will return a valid pointer to this interface to the client whenever the latter queries the property *CurveArray* (cf. 2.3.1.3.7) which is part of the interface *IRealtimeDisplay* (cf. 2.3.1).

2.3.4.1.5 The property *CurveArray* (cf. 2.5.4.3.3) accessible via this interface consists of three arrays of values which describe a three-dimensional breakpoint table.

2.3.4.1.6 The first of these three arrays, which contains the fixed parameters of the 3d breakpoint table that is to be displayed as a projection to a two-dimensional plot and, thus, forms an array of curves. Hereby, the size of the parameter array is equal to the number of curves. This array shall be defined in the form of `parameters[i]`, with `i = 0,.1,..,n-1` (`n` curves).

2.3.4.1.7 Both the arrays containing the variable values are two-dimensional arrays of the form `x[i][j]` and `y[i][j]`, respectively. While `i` is denoting the curve index (cf. 2.3.4.1.6), `j` respresents the zero based point index of the curve with index `i`. The number of points may be different for each curve.

2.3.4.1.8 For being displayed in the diagram of a RTD Y/X Plot ActiveX Control, the precondition for the values of the array `x` is: `x[i][j] < x[i][j+1]` for all indices `i`, i.e. for all curves.

2.3.4.1.9 Design

2.3.4.1.10 This interface will be a dispatch interface.

2.3.4.1.11 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.3.4.1.12 The ProgID for this interface is *"CurveArrayCtl.CurveArray"*.

2.3.4.2 Methods and Properties

2.3.4.2.1 Property *ParameterName*

```
[propget, id(1), helpstring("property ParameterName")] HRESULT
ParameterName([out, retval] BSTR *pVal);
[propput, id(1), helpstring("property ParameterName")] HRESULT
ParameterName([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	The name of the fixed parameter for each curve of the array.

2.3.4.2.2 Property *ParameterEngUnit*

```
[propget, id(2), helpstring("property ParameterEngUnit")] HRESULT
ParameterEngUnit([out, retval] BSTR *pVal);
[propput, id(2), helpstring("property ParameterEngUnit")] HRESULT
ParameterEngUnit([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	The engineering unit of the fixed parameter for each curve of the array.

2.3.4.2.3 Property *ParameterDescription*

```
[propget, id(3), helpstring("property ParameterDescription")] HRESULT
ParameterDescription([out, retval] BSTR *pVal);
[propput, id(3), helpstring("property ParameterDescription")] HRESULT
ParameterDescription([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	The description of the fixed parameter for each curve of the array.

2.3.4.2.4 Property *CurveCount*

```
[propget, id(4), helpstring("property CurveCount")] HRESULT
CurveCount([out, retval] long *pVal);
```

Argument Name	Description
*pVal	Return the number of curves part of the array.

The value of this property is needed for a query of the property *CurveParameters* (cf. 2.3.4.2.5).

2.3.4.2.5 Property *CurveParameters*

```
[propget, id(5), helpstring("property CurveParameters")] HRESULT
CurveParameters([in] long lCount,[out, retval, size_is(lCount)] double
*pVal);
[propput, id(5), helpstring("property CurveParameters")] HRESULT
CurveParameters([in] long lCount, [in, size_is(lCount)] double
*newVal);
```

Argument Name	Description
`lCount`	The number of curves. In case of a query of this property, `lCount` has to be derived from the property *CurveCount* (cf. 2.3.4.2.4) first.
`*pVal, *newVal`	The array which contains the fixed parameters for all curves.

2.3.4.2.6 Property <*coordinate*>*Name*

```
[propget, id(6), helpstring("property XName")] HRESULT XName(
[out, retval] BSTR *pVal);
[propput, id(6), helpstring("property XName")] HRESULT XName(
[in] BSTR newVal);
[propget, id(7), helpstring("property YName")] HRESULT YName(
[out, retval] BSTR *pVal);
[propput, id(7), helpstring("property YName")] HRESULT YName(
[in] BSTR newVal);
```

Argument Name	Description
`*pVal, newVal`	The name of the x or y variable, respectively.

2.3.4.2.7 Property <*coordinate*>*EngUnit*

```
[propget, id(8), helpstring("property XEngUnit")] HRESULT
XEngUnit([out, retval] BSTR *pVal);
[propput, id(8), helpstring("property XEngUnit")] HRESULT
XEngUnit([in] BSTR newVal);
[propget, id(9), helpstring("property YEngUnit")] HRESULT
YEngUnit([out, retval] BSTR *pVal);
[propput, id(9), helpstring("property YEngUnit")] HRESULT
YEngUnit([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	The string containing the engineering unit of the x or y variable, respectively.

2.3.4.2.8 Property <*coordinate*>*Description*

```
[propget, id(10), helpstring("property XDescription")] HRESULT
XDescription([out, retval] BSTR *pVal);
[propput, id(10), helpstring("property XDescription")] HRESULT
XDescription([in] BSTR newVal);
[propget, id(11), helpstring("property YDescription")] HRESULT
YDescription([out, retval] BSTR *pVal);
[propput, id(11), helpstring("property YDescription")] HRESULT
YDescription([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	The description of the x or y variable, respectively.

2.3.4.2.9 Property <*coordinate*>*DisplayMin*

```
[propget, id(12), helpstring("property XDisplayMin")] HRESULT
XDisplayMin([out, retval] double *pVal);
[propput, id(12), helpstring("property XDisplayMin")] HRESULT
XDisplayMin([in] double newVal);
[propget, id(13), helpstring("property YDisplayMin")] HRESULT
YDisplayMin([out, retval] double *pVal);
[propput, id(13), helpstring("property YDisplayMin")] HRESULT
YDisplayMin([in] double newVal);
```

Argument Name	Description
*pVal, newVal	The minimum display value of the x or y variable, respectively.

2.3.4.2.10 Property <*coordinate*>*DisplayMax*

```
[propget, id(14), helpstring("property XDisplayMax")] HRESULT
XDisplayMax([out, retval] double *pVal);
[propput, id(14), helpstring("property XDisplayMax")] HRESULT
XDisplayMax([in] double newVal);
[propget, id(15), helpstring("property YDisplayMax")] HRESULT
YDisplayMax([out, retval] double *pVal);
[propput, id(15), helpstring("property YDisplayMax")] HRESULT
YDisplayMax([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The maximum display value of the x or y variable, respectively.

2.3.4.2.11 Method *put_CurvePoints*

```
[id(16), helpstring("method put_CurvePoints")] HRESULT
put_CurvePoints(
[in] long CurveIndex,
[in] long PointCount,
[in, size_is(PointCount)] double *xVals,
[in, size_is(PointCount)] double *yVals);
```

Argument Name	Description
`CurveIndex`	The zero based index of the curve of which the point co-ordinates are going to be set.
`PointCount`	The number of points of the curve with index `CurveIndex`.
`*xVals, *yVals`	The arrays which contain the values for the x and y co-ordinates, respectively, of this curve.

Note that the sequence of the values of both arrays has to be such that `xVals[CurveIndex][PointIndex] < xVals[CurveIndex][PointIndex+1]` is fulfilled for all curves.

2.3.4.2.12 Property *CurvePointCount*

```
[propget, id(17), helpstring("property CurvePointCount")] HRESULT
CurvePointCount([in] long CurveIndex, [out, retval] long *pVal);
```

Argument Name	Description
`CurveIndex`	The zero based index of the curve of which the number of points has to be retrieved.
`*pVal`	Return the number of points of the curve with index `CurveIndex`.

2.3.4.2.13 Method *get_CurvePoints*

```
[id(18), helpstring("method put_CurvePoints")] HRESULT
put_CurvePoints(
[in] long CurveIndex,
[in] long PointCount,
[out, size_is(PointCount)] double *xVals,
```

```
[out, size_is(PointCount)] double *yVals);
```

Argument Name	Description
`CurveIndex`	The zero based index of the curve of which the point co-ordinates are to be retrieved.
`PointCount`	The number of points of the curve with index `CurveIndex`. This parameter must be equal to the value of the property *CurvePointCount* (cf. 2.3.4.2.12) and, therefore, has to be retrieved from the latter first.
`*xVals, *yVals`	The arrays which contain the values for the x and y co-ordinates, respectively. Note that both these arrays have to be provided with the size `PointCount` by the client.

2.3.4.2.14 Method *IsModified*

```
[id(19), helpstring("method IsModified")] HRESULT IsModified([out,
retval] BOOL *pVal);
```

Argument Name	Description
`*pVal`	Return flag indicating whether (`TRUE`) or not (`FALSE`) a property of the *CurveArray* object has been modified by the client.

2.3.4.2.15 Method *ClearModifiedFlag*

```
[id(20), helpstring("method ClearModifiedFlag")] HRESULT
ClearModifiedFlag();
```

Whenever a client calls this method, the modified flag which can be retrieved via the property *IsModified* (cf. 2.3.4.2.14) is set to `FALSE`.

2.3.4.2.16 Method *Undefine*

```
[id(21), helpstring("method Undefine")] HRESULT Undefine();
```

Whenever this method is called either directly or via the method *UndefineRTDVars* (cf. 2.3.1.3.12), the defined flag of the CurveArray object is set to `FALSE`. This indicates to the RTD ActiveX Control not to display any data of this object at all.

2.3.4.2.17 Method *IsDefined*

```
[id(22), helpstring("method IsDefined")] HRESULT IsDefined([out,
retval] BOOL *pVal);
```

Argument Name	Description
*pVal	Return flag indicating whether (TRUE) or not (FALSE) the CurveArray object is defined. This flag is automatically set to TRUE whenever a property of the object is modified, and to FALSE by a call of the method *Undefine* (cf. 2.3.4.2.16).

This method is needed by the RTD ActiveX Control to determine whether the define flag of the CurveArray object is set to TRUE or FALSE.

2.3.4.2.18 Property *DispID*

```
[propget, id(23), helpstring("property DispID")] HRESULT DispID(
[out, retval] DISPID *pVal);
[propput, id(23), helpstring("property DispID")] HRESULT DispID(
[in] DISPID newVal);
```

Argument Name	Description
*pVal, newVal	Returns the DISPID of the variable designated to be mapped to a CurveArray object.

2.3.4.2.18.1 Note that the RTD ActiveX Control, which has this interface implemented, will initially set the value of this property equal to the DISPID (cf. 2.5.4.3.3) of its variable that is designated to be mapped to a client variable representing a CurveArray object, i.e. a 3d breakpoint table.

2.3.4.2.18.2 This property has to be queried by the RTD client, to determine whether or not the CurveArray object has been mapped to the designated property of the RTD ActiveX Control.

2.3.4.3 Events Fired

There are no events fired by this interface.

2.3.4.4 Usage Conditions and Restrictions

2.3.4.4.1 This interface is accessed by the RTD client via a query of the property *CurveArray* (cf. 2.3.1.3.7) which is part of the interface *IRealtimeDisplay* (cf. 2.3.1). This property returns a pointer to this interface.

2.3.4.4.2 Note that only the RTD Y/X Plot ActiveX Control will deliver a valid address for this interface, i.e. no other RTD ActiveX Control will have it implemented.

2.3.4.5 Persistent Data

There is no data stored persistently by this interface.

2.3.4.6 Example

2.3.4.6.1 Query the interface *ICurveArray* from a RTD ActiveX Control and set some properties of the 3d breakpoint table object

The following example illustrates the access of the interface *ICurveArray* via the the interface *IRealtimeDisplay* by a Visual C++ client, i.e. the RTDE.

2.3.4.6.1.1 The RTD client checks first whether it deals with a RTD ActiveX Control:

```
IRealtimeDisplayPtr RealtimeDisplayPtr (GetControlLPDISPTACH());
if (RealtimeDisplayPtr == NULL)
   return;    // not a RTD ActiveX Control
```

2.3.4.6.1.2 Set the defined flag of all RTD Variables including the corresponding property of the 3d breakpoint table object to FALSE. Note that calling this method is important after the mapping process has been finished, because otherwise the RTD ActiveX Control would still display the data of the control's variables that have been disconnected from client variables and, thus, have their modification flag set to `TRUE`.

```
RealtimeDisplayPtr -> UndefineRTDVars();
```

2.3.4.6.1.3 Get interface *ICurveArray*:

```
ICurveArrayPtr CurveArrayPtr (RealtimeDisplayPtr ->
                              GetCurveArray());
If (CurveArrayPtr == NULL)
   return;    // There is no such object contained in the control
```

2.3.4.6.1.4 Check whether a CurveArray (i.e. a 3d breakpoint table) object has been mapped to :

```
// channels etc. directly mapped to the control using
// AXProperties
for (long i = 1; i <= AXProperties.GetCount(); i++)
{
   CDVAXProperty AXProperty = AXProperties.Item(COleVariant(i));

   const DISPID dispID = AXProperty.GetDispid();

   if (CurveArrayPtr != NULL &&
       dispID == CurveArrayPtr->GetDispID())
   {
      //  The 3d breakpoint table object has been mapped
      CurveArrayPtr->PutParameterName(<some String>);
```

```
            CurveArrayPtr->PutxName(<some String>);
            CurveArrayPtr->PutyName(<some String>);
               :          :       :     :
               :          :       :     :
            CurveArrayPtr->PutxDisplayMin(<some double number>);
            CurveArrayPtr->PutxDisplayMax(<some double number>);
            CurveArrayPtr->PutyDisplayMin(<some double number>);
            CurveArrayPtr->PutyDisplayMax(<some double number>);
               :          :       :     :
               :          :       :     :
      }
}
```

2.4 Overview of the general interfaces offered by the different RTD ActiveX Controls

2.4.1 This section gives an overview of which of the general interfaces are implemented in the different RTD ActiveX Controls and, if so, how many instances of it are provided.

2.4.2 The following table summarises the most important features of the various RTD ActiveX Controls and gives an overview of which of the general interfaces (cf. 2.3) are implemented in each control as well as what are the names of the individual, control specific interfaces:

RTD ActiveX Control	Graph	IFullset	ICurveArray	Buffered Data	Number of RTDVars to be mapped	Monitored Variables	Name of specific interface	ProgID of the interface
RTD Time	—	—	—	—	2	0	*IRTDTime*	RTDTimeCtl.RTDTime
RTD Value	—	—	—	—	1	1 (fixed)	*IRTDValue*	RTDValueCtl.RTDValue
RTD Y/X Plot	√	√	√	√	20	0	*IRTDYXPlot*	RTDYXPlotCtl.RTDYXPlot
RTD Profile Plot	√	√	—	√	100	0	*IRTDProfilePlot*	RTDProfilePlotCtl.RTDProfilePlot
RTD Strip Chart	√	—	—	√	10	0 or 1 selectable	*IRTDStripChart*	RTDStripChartCtl.RTDStripChart
RTD Input	—	—	—	—	1	0	*IRTDInput*	RTDInputCtl.RTDInput

Table 1: Overview of the general interfaces implemented in the different RTD ActiveX Controls.

2.5 Specific interfaces

The following paragraphs of this section describe the interfaces specific to the individual RTD ActiveX Controls. These interfaces are used to make certain RTD Variables of the control public via their respective pre-defined DISPID and, thus, connectable to a client variable of either type float or string.

2.5.1 General Layout Properties

The properties listed in the following subparagraphs of this section are common to all RTD ActiveX Controls and made public via their pre-defined DISPIDs. Since they are all related to colour settings, the stock colour property page with the class identifier (CLSID) `CLSID_StockColorPage` is designated to be used for accessing them. However, this requires that they have to be defined in each of the control specific interfaces, which are described in the subsequent sections (cf. 2.5.2 to 2.5.7).

2.5.1.1 Property *BackgroundColor*

```
// DispID for background color
const DISPID DISPID_BKCOLOR = 251;

[propget, id(DISPID_BKCOLOR), helpstring("property BackgroundColor")]
HRESULT BackgroundColor ([out, retval] OLE_COLOR *pVal);
[propput, id(DISPID_BKCOLOR), helpstring("property BackgroundColor")]
HRESULT BackgroundColor ([in] OLE_COLOR newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_BKCOLOR`

This property of type OLE_COLOR represents the background colour of the drawing area of the RTD ActiveX Control outside of the numerical display field.

2.5.1.2 Property *NumericBkColor*

```
// DispID for background color of the numerical output field
const DISPID DISPID_NUMERIC_BKCOLOR = 252;

[propget, id(DISPID_NUMERIC_BKCOLOR), helpstring("property
NumericBkColor")] HRESULT NumericBkColor ([out, retval] OLE_COLOR
*pVal);
[propput, id(DISPID_NUMERIC_BKCOLOR), helpstring("property
NumericBkColor")] HRESULT NumericBkColor ([in] OLE_COLOR newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_NUMERIC_BKCOLOR`

This property of type OLE_COLOR represents the background colour of the numerical display field.

2.5.1.3 Property *FontColor*

```
// DispID for font color of the name, description, and engineering
unit
```

```
const DISPID DISPID_FONTCOLOR = 253;

[propget, id(DISPID_FONTCOLOR), helpstring("property FontColor")]
HRESULT FontColor ([out, retval] OLE_COLOR *pVal);
[propput, id(DISPID_FONTCOLOR), helpstring("property FontColor")]
HRESULT FontColor ([in] OLE_COLOR newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_FONTCOLOR`

This property of type OLE_COLOR represents the font colour of the strings representing all the static information about the RTD Variables to be displayed (e.g. their names and engineering units).

2.5.1.4 Property *NumericFontColor*

```
// DispID for font color of the numerical output
const DISPID DISPID_NUMERIC_FONTCOLOR = 254;

[propget, id(DISPID_NUMERIC_FONTCOLOR), helpstring("property
NumericFontColor")] HRESULT NumericFontColor ([out, retval] OLE_COLOR
*pVal);
[propput, id(DISPID_NUMERIC_FONTCOLOR), helpstring("property
NumericFontColor")] HRESULT NumericFontColor ([in] OLE_COLOR newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_NUMERICFONTCOLOR`

This property of type OLE_COLOR represents the font colour of the numerical output string.

2.5.2 Interface *IRTDTime*

2.5.2.1 Introduction

2.5.2.1.1 This interface is specific to the RTD Time Display ActiveX Control.

2.5.2.1.2 The RTD Time Display ActiveX Control provides two RTD Variables designated to be mapped to client variables of the format enumeration value `Date` and `Time` (cf. 2.3.2.3.1), respectively.

2.5.2.1.3 These RTD Variables are accessible via the properties *Date* (cf. 2.5.2.3.1) and *Time* (cf. 2.5.2.3.2) of this interface.

2.5.2.2 Design

2.5.2.2.1 This interface will be a dispatch interface.

2.5.2.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.2.2.3 The ProgID for this interface is *"RTDTimeCtl.RTDTime"*.

2.5.2.3 Methods and Properties

2.5.2.3.1 Property *Date*

```
const DISPID DISPID_DATE = 101;

[propget, id(DISPID_DATE), helpstring("property Date")] HRESULT
Date([out, retval] double *pVal);
[propput, id(DISPID_DATE), helpstring("property Date")] HRESULT
Date([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_DATE`.

2.5.2.3.2 Property *Time*

```
const DISPID DISPID_TIME = 102;

[propget, id(DISPID_TIME), helpstring("property Time")] HRESULT
Time([out, retval] double *pVal);
[propput, id(DISPID_TIME), helpstring("property Time")] HRESULT
Time([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_TIME`.

2.5.2.4 Events Fired

There are no events fired by this interface.

2.5.2.5 Usage Conditions and Restrictions

There are no special conditions or restrictions concerning the usage of this interface.

2.5.2.6 Persistent Data

2.5.2.6.1 This interface initiates that the two RTDVar objects (cf. 2.5.2.3.1 and 2.5.2.3.2, respectively) mapped to client variables are stored persistently by their respective interface *IRTDVar* (cf. 2.3.2).

2.5.2.6.2 In addition, all other properties which can be modified via the property dialogue of the RTD Date & Time Display ActiveX Control are stored persistently by this interface.

2.5.3 Interface *IRTDValue*

2.5.3.1 Introduction

2.5.3.1.1 This interface is specific to the RTD Value ActiveX Control.

2.5.3.1.2 The RTD Value ActiveX Control provides a single RTD Variable designated to be mapped to an online channel of the client.

2.5.3.1.3 This RTD Variable is accessible via the property *Value* (cf. 2.5.3.3.1) of this interface.

2.5.3.2 Design

2.5.3.2.1 This interface will be a dispatch interface.

2.5.3.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.3.2.3 The ProgID for this interface is *"RTDValueCtl.RTDValue"*.

2.5.3.3 Methods and Properties

2.5.3.3.1 Property *Value*

```
// DispID for the variable to be displayed
const DISPID DISPID_VALUE1 = 101;

[propget, id(DISPID_VALUE1), helpstring("property Value")] HRESULT
Value([out, retval] double *pVal);
[propput, id(DISPID_VALUE1), helpstring("property Value")] HRESULT
Value([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_VALUE1`

2.5.3.3.1.1 In addition to this RTD Variable, which is made public with the DISPID `DISPID_VALUE1` in the property map of the default project class of the RTD Value Display ActiveX Control reserved for being mapped to a client variable (online channel), there are four other RTDVar objects defined.

2.5.3.3.1.2 In case that the client variable mapped to the RTD Variable with DISPID `DISPID_VALUE1` is monitored by others, the client application is able to attribute up to four monitoring variables (i.e. channels) to those additional RTDVar objects.

2.5.3.3.2 Property *BarBkColor*

```
// DispID for the variable to be displayed
const DISPID DISPID_BAR_BKCOLOR = 255;

[propget, id(DISPID_BAR_BKCOLOR), helpstring("property BarBkColor")]
HRESULT BarBkColor([out, retval] OLE_COLOR *pVal);
[propput, id(DISPID_BAR_BKCOLOR), helpstring("property BarBkColor")]
HRESULT BarBkColor([in] OLE_COLOR newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the property with DISPID `DISPID_BAR_BKCOLOR`

This property of type OLE_COLOR specifies the background colour for the bar display field of the RTD Value Display ActiveX Control. Together with the other general properties all related to colour settings (cf. 2.5.1), this property is set via the stock colour property page with the CLSID `CLSID_StockColorPage`.

2.5.3.4 Events Fired

There are no events fired by this interface.

2.5.3.5 Usage Conditions and Restrictions

2.5.3.5.1 There are no special conditions or restrictions concerning the usage of this interface.

2.5.3.6 Persistent Data

2.5.3.6.1 This interface initiates that the single RTDVar object (cf. 2.5.3.3.1) mapped to the client variable is stored persistently by its respective interface *IRTDVar* (cf. 2.3.2).

2.5.3.6.2 In addition, all other properties which can be modified via the property dialogue of the RTD Value Display ActiveX Control are stored persistently by this interface.

2.5.4 Interface *IRTDYXPlot*

2.5.4.1 Introduction

2.5.4.1.1 This interface is specific to the RTD Y/X Plot ActiveX Control.

2.5.4.1.2 The RTD Y/X Plot ActiveX Control has to provide twenty RTD Variables (ten for x and y co-ordinates, respectively) designated to be mapped to online channels of the client.

2.5.4.1.3 These RTD Variables are accessible via the properties *XValue<index>* and *YValue<index>*, respectively, of this interface.

2.5.4.2 Design

2.5.4.2.1 This interface will be a dispatch interface.

2.5.4.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.4.2.3 The ProgID for this interface is *"RTDYXPlotCtl.RTDYXPlot"*.

2.5.4.3 Methods and properties

2.5.4.3.1 Property *XValue<index>*

```
const DISPID DISPID_XVALUE01 = 101;
const DISPID DISPID_XVALUE02 = 102;
  :     :      :      :      :  :
  :     :      :      :      :  :
const DISPID DISPID_XVALUE10 = 110;

[propget, id(DISPID_XVALUE01), helpstring("property XValue01")]
HRESULT XValue01([out, retval] double *pVal);
[propput, id(DISPID_XVALUE01), helpstring("property XValue01")]
HRESULT XValue01([in] double newVal);
            :           :                 :           :
            :           :                 :           :
[propget, id(DISPID_XVALUE10), helpstring("property XValue10")]
HRESULT XValue10([out, retval] double *pVal);
[propput, id(DISPID_XVALUE10), helpstring("property XValue10")]
HRESULT XValue10([in] double newVal);
```

Argument Name	Description
*pVal, newVal	The value of the RTD variable with the DISPID DISPID_XVALUE<index>

If the client does not map any of its variables to such an x variable, the RTD Y/X Plot ActiveX Control will display one x item with default properties. This will form a single curve with the corresponding y item (cf. 2.5.4.3.2).

2.5.4.3.2 Property *YValue<index>*

```
const DISPID DISPID_YVALUE01 = 151;
const DISPID DISPID_YVALUE02 = 152;
  :     :      :       :     :  :
  :     :      :       :     :  :
const DISPID DISPID_YVALUE10 = 160;

[propget, id(DISPID_YVALUE01), helpstring("property YValue01")]
HRESULT YValue01([out, retval] double *pVal);
[propput, id(DISPID_YVALUE01), helpstring("property YValue01")]
HRESULT YValue01([in] double newVal);
            :           :                 :           :
            :           :                 :           :
[propget, id(DISPID_YVALUE10), helpstring("property YValue10")]
HRESULT YValue10([out, retval] double *pVal);
[propput, id(DISPID_YVALUE10), helpstring("property YValue10")]
HRESULT YValue10([in] double newVal);
```

Argument Name	Description
*pVal, newVal	The value of the RTD variable with the DISPID DISPID_YVALUE<index>

If the client does not map any of its variables to such a y variable, the RTD Y/X Plot ActiveX Control will display one y item with some default properties. This will form a single curve with the corresponding x item (cf. 2.5.4.3.1).

2.5.4.3.3 Property *CurveArray*

```
const DISPID DISPID_CURVEARRAY = 270;

[propget, id(DISPID_CURVEARRAY), helpstring("property CurveArray")]
HRESULT CurveArray([out, retval] BSTR *pVal);
[propput, id(DISPID_CURVEARRAY), helpstring("property CurveArray")]
HRESULT CurveArray([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	The name of the RTD variable with the DISPID DISPID_CURVEARRAY

This property is designated to be mapped to a CurveArray object, such as a three-dimensional breakpoint table.

2.5.4.4 Events Fired

There are no events fired by this interface.

2.5.4.5 Usage Conditions and Restrictions

2.5.4.5.1 This interface has to be used first by the client to map his variables to the RTD variables of the RTD Y/X Plot ActiveX Control.

2.5.4.5.2 In a subsequent step, the interface (i.e. *IRealtimeDisplay*) and the property sheet of the control may be used to retrieve further information from the control and to set additional properties, respectively.

2.5.4.6 Persistent Data

2.5.4.6.1 This interface initiates that the RTDVar objects (cf. 2.5.4.3.1 and 2.5.4.3.2) mapped to client variables are stored persistently by their respective interface *IRTDVar* (cf. 2.3.2).

2.5.4.6.2 In addition, all other properties provided by this interface and also those which can be modified via the property dialogue of the RTD Y/X Plot ActiveX Control are stored persistently by this interface.

2.5.5 Interface *IRTDProfilePlot*

2.5.5.1 Introduction

2.5.5.1.1 This interface is specific to the RTD Profile Plot ActiveX Control.

2.5.5.1.2 The RTD Profile Plot ActiveX Control has to provide one hundred RTD Variables (fifty for x and y co-ordinates) designated to be mapped to online channels of the client.

2.5.5.1.3 These RTD Variables are accessible via the properties *XValue<index>* and *YValue<index>*, respectively, of this interface.

2.5.5.2 Design

2.5.5.2.1 This interface will be a dispatch interface.

2.5.5.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.5.2.3 The ProgID for this interface is *"RTDProfilePlotCtl.RTDProfilePlot"*.

2.5.5.3 Methods and properties

2.5.5.3.1 Property *XValue<index>*

```
const DISPID DISPID_XVALUE01 = 101;
const DISPID DISPID_XVALUE02 = 103;
  :    :       :      :     :  :
  :    :       :      :     :  :
const DISPID DISPID_XVALUE50 = 199;

[propget, id(DISPID_XVALUE01), helpstring("property XValue01")]
HRESULT XValue01([out, retval] double *pVal);
[propput, id(DISPID_XVALUE01), helpstring("property XValue01")]
HRESULT XValue01([in] double newVal);
            :           :                 :           :
            :           :                 :           :
[propget, id(DISPID_XVALUE01), helpstring("property XValue50")]
HRESULT XValue50([out, retval] double *pVal);
[propput, id(DISPID_XVALUE50), helpstring("property XValue50")]
HRESULT XValue50([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the RTD variable with the DISPID `DISPID_XVALUE<index>`

If the client does not map any of its variables to such an x variable, the RTD Profile Plot ActiveX Control will display one x item with its default properties. This will form a single curve with the corresponding y item (cf. 2.5.5.3.2).

2.5.5.3.2 Property *YValue<index>*

```
const DISPID DISPID_YVALUE01 = 102;
const DISPID DISPID_YVALUE02 = 104;
  :      :      :      :     :  :
  :      :      :      :     :  :
const DISPID DISPID_YVALUE50 = 200;

[propget, id(DISPID_YVALUE01), helpstring("property YValue01")]
HRESULT YValue01([out, retval] double *pVal);
[propput, id(DISPID_YVALUE01), helpstring("property YValue01")]
HRESULT YValue01([in] double newVal);
            :           :                 :           :
            :           :                 :           :
[propput, id(DISPID_YVALUE50), helpstring("property YValue50")]
HRESULT YValue50([out, retval] double *pVal);
[propput, id(DISPID_YVALUE50), helpstring("property YValue50")]
HRESULT YValue50([in] double newVal);
```

Argument Name	Description

`*pVal, newVal`	The value of the RTD variable with the DISPID `DISPID_YVALUE<index>`

If the client does not map any of its variables to such a y variable, the RTD Profile Plot ActiveX Control will display one y item with its default properties. This will form a single curve with the corresponding x item (cf. 2.5.5.3.1).

2.5.5.4 Events Fired

There are no events fired by this interface.

2.5.5.5 Usage Conditions and Restrictions

2.5.5.5.1 This interface has to be used first by the user to map his variables to the RTD variables of the RTD Profile Plot ActiveX Control.

2.5.5.5.2 In a subsequent step, the interface *IRealtimeDisplay* (cf. 2.3.1) and the property sheet of the control may be used by the RTD client and the interactive user, respectively, to retrieve further information from the control, transfer data and to set further properties.

2.5.5.6 Persistent Data

2.5.5.6.1 This interface initiates that the RTDVar objects (cf. 2.5.5.3.1 and 2.5.5.3.2) mapped to client variables are stored persistently by their respective interface *IRTDVar* (cf. 2.3.2).

2.5.5.6.2 In addition, all other properties provided by this interface and also those which can be modified via the property dialogue of the RTD Profile Plot ActiveX Control are stored persistently by this interface.

2.5.6 Interface *IRTDStripChart*

2.5.6.1 Introduction

2.5.6.1.1 This interface is specific to the RTD Strip Chart ActiveX Control.

2.5.6.1.2 The RTD Strip Chart ActiveX Control has to provide ten RTD Variables designated to be mapped to online channels of the client.

2.5.6.1.3 These RTD Variables are accessible via the properties *Value<index>* of this interface (cf. 2.5.6.3.1).

2.5.6.1.3.1 There are four additional RTD Variables reserved by a RTD Strip Chart ActiveX Control, if one of the ten RTD Variables is mapped to an online channel which can be monitored by up to four limiting channels.

2.5.6.1.3.2 An interactive user can select one of those online variables as being monitored in the property page of the RTD Strip Chart ActiveX Control.

2.5.6.2 Design

2.5.6.2.1 This interface will be a dispatch interface.

2.5.6.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.6.2.3 The ProgID for this interface is *"RTDStripChartCtl.RTDStripChart"*.

2.5.6.3 Methods and properties

2.5.6.3.1 Property *Value<index>*

```
const DISPID DISPID_VALUE01 = 101;
const DISPID DISPID_VALUE02 = 102;
  :      :      :     :     :  :
  :      :      :     :     :  :
const DISPID DISPID_VALUE10 = 110;

[propget, id(DISPID_VALUE01), helpstring("property Value01")] HRESULT
Value01([out, retval] double *pVal);
[propput, id(DISPID_VALUE01), helpstring("property Value01")] HRESULT
Value01([in] double newVal);
            :           :                :           :
            :           :                :           :
[propget, id(DISPID_VALUE10), helpstring("property Value10")] HRESULT
Value10([out, retval] double *pVal);
[propput, id(DISPID_VALUE10), helpstring("property Value10")] HRESULT
Value10([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value the RTD variable with the DISPID `DISPID_VALUE<index>`

If the user does not map any of its variables to such a variable, the RTD Strip Chart ActiveX Control will display one y item with default properties. This will form a single curve displaying the values of this variable as a function of time.

2.5.6.4 Events Fired

There are no events fired by this interface.

2.5.6.5 Usage Conditions and Restrictions

2.5.6.5.1 This interface has to be used first by the client to map his variables to the RTD Variables of the RTD Strip Chart ActiveX Control.

2.5.6.5.2 In a subsequent step, the interface (i.e. *IRealtimeDisplay*) and the property sheet of the control may be used to retrieve further information from the control and to set additional properties, respectively.

2.5.6.6 Persistent Data

2.5.6.6.1 This interface initiates that the RTDVar objects (cf. 2.5.6.3.1) mapped to client variables are stored persistently by their respective interface *IRTDVar* (cf. 2.3.2).

2.5.6.6.2 All other properties provided by this interface and also those which can be modified via the property dialogue of the RTD Strip Chart ActiveX Control are stored persistently by this interface.

2.5.6.6.3 In addition, the property *MonitoredDispID* (cf. 2.3.1.3.5) of the interface *IRealtimeDisplay* is stored persistently by this interface.

2.5.7 Interface *IRTDInput*

2.5.7.1 Introduction

2.5.7.1.1 This interface is specific to the RTD Input ActiveX Control.

2.5.7.1.2 The RTD Input ActiveX Control provides one RTD Variable designated to be mapped to a client channel of format type (cf. 2.3.2.3.13) integer, Boolean, or float.

2.5.7.1.3 This RTD Variable is accessible via the property *Value* (cf. 2.5.7.3.1) of this interface.

2.5.7.2 Design

2.5.7.2.1 This interface will be a dispatch interface.

2.5.7.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.7.2.3 The ProgID for this interface is *"RTDInputCtl.RTDInput"*.

2.5.7.3 Methods and Properties

2.5.7.3.1 Property *Value*

```
// DISPID for the variable to be displayed
const DISPID DISPID_VALUE1 = 101;
```

```
[propget, id(DISPID_VALUE1), helpstring("property Value")] HRESULT
Value([out, retval] double *pVal);
[propput, id(DISPID_VALUE1), helpstring("property Value")] HRESULT
Value([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the `one and only RTD variable associated with the` RTD Input ActiveX Control

2.5.7.4 Events Fired

There are no events fired by this interface.

2.5.7.5 Usage Conditions and Restrictions

2.5.7.5.1 The RTD Input ActiveX Control is used to enable an interactive user either to select a specific value from a list of discrete values or to insert a new float number as value for a certain client variable mapped to the control's RTD Variable.

2.5.7.5.2 First, this interface has to be used by an interactive user to map a variable (client channel) to the RTD Variable of the control with the property *Value* (cf. 2.5.7.3.1).

2.5.7.5.3 In case of a variable of format type integer, a list of discrete values may be provided as strings by the client application via the property *Format* (cf. 2.3.2.3.14), which is part of the interface *IRTDVar* (cf. 2.3.1.3.6).

2.5.7.5.4 The client application is able to set all properties of this mapped variable via the property *RTDVar* (cf. 2.3.1.3.6) part of the control's interface *IRealtimeDisplay*.

2.5.7.5.5 During runtime, the RTDD will set an initial value for the variable when the control is first displayed.

2.5.7.5.6 Later on, if the value of the property *InputVariable* (cf. 2.3.2.3.23), which is part of the interface *IRTDVar* (cf. 2.3.2), is set to `TRUE`, the RTD Input ActiveX Control will allow an interactive user to modify the value of the channel mapped to its RTD Variable.

2.5.7.5.7 Then, the RTD Input ActiveX Control will fire the new value with the event *ChangeValue* (cf. 2.3.1.4.5) to the RTD client, after the interactive user has confirmed the selection of another value for that variable.

2.5.7.6 Persistent Data

2.5.7.6.1 This interface initiates that the single RTDVar object (cf. 2.5.7.3.1) mapped to a client variable is stored persistently by its respective interface *IRTDVar* (cf. 2.3.2).

2.5.7.6.2 All other properties provided by this interface and also those which can be modified via the property dialogue of the RTD Input ActiveX Control are stored persistently by this interface.

2.5.8 Interface *IRTDMultiSwitch*

2.5.8.1 Introduction

2.5.8.1.1 This interface is specific to the RTD Multi Switch ActiveX Control.

2.5.8.1.2 The RTD Multi Switch ActiveX Control shall allow an interactive user to set new Boolean values for multiple Boolean client variables similar to using a radio group.

2.5.8.1.3 The RTD Multi Switch ActiveX Control provides five RTD Variables designated to be mapped to Boolean channels.

2.5.8.1.4 These RTD Variables are accessible via the *Value* properties (cf. 2.5.8.3.1) of this interface.

2.5.8.2 Design

2.5.8.2.1 This interface will be a dispatch interface.

2.5.8.2.2 This interface will be a dual interface, i.e. it supports automation clients as well as C++ and VB clients.

2.5.8.2.3 The ProgID for this interface is *"RTDMultiSwitchCtl.RTDMultiSwitch"*.

2.5.8.3 Methods and Properties

2.5.8.3.1 Property *Value<index>*

```
// DISPID for the variables to be displayed and manipulated
const DISPID DISPID_VALUE1 = 101;
const DISPID DISPID_VALUE2 = 102;
const DISPID DISPID_VALUE3 = 103;
const DISPID DISPID_VALUE4 = 104;
const DISPID DISPID_VALUE5 = 105;

[propget, id(DISPID_VALUE1), helpstring("property Value1")] HRESULT
Value1([out, retval] double *pVal);
[propput, id(DISPID_VALUE1), helpstring("property Value1")] HRESULT
Value1([in] double newVal);
[propget, id(DISPID_VALUE2), helpstring("property Value2")] HRESULT
Value2([out, retval] double *pVal);
[propput, id(DISPID_VALUE2), helpstring("property Value2")] HRESULT
Value2([in] double newVal);
```

```
[propget, id(DISPID_VALUE3), helpstring("property Value3")] HRESULT
Value3([out, retval] double *pVal);
[propput, id(DISPID_VALUE3), helpstring("property Value3")] HRESULT
Value3([in] double newVal);
[propget, id(DISPID_VALUE4), helpstring("property Value4")] HRESULT
Value4([out, retval] double *pVal);
[propput, id(DISPID_VALUE4), helpstring("property Value4")] HRESULT
Value4([in] double newVal);
[propget, id(DISPID_VALUE5), helpstring("property Value5")] HRESULT
Value5([out, retval] double *pVal);
[propput, id(DISPID_VALUE5), helpstring("property Value5")] HRESULT
Value5([in] double newVal);
```

Argument Name	Description
`*pVal, newVal`	The value of the RTD variable associated with the respective DISPID `DISPID_VALUE`

2.5.8.4 Events Fired

There are no events fired by this interface.

2.5.8.5 Usage Conditions and Restrictions

2.5.8.5.1 The RTD Multi Switch ActiveX Control is used to enable an interactive user to set new Boolean values for multiple client variables mapped to the control's RTD Variables similar to using a radio group.

2.5.8.5.2 First, this interface has to be used by an interactive user to map one or multiple variables (client channels) to the RTD Variables of the control with the *Value* properties (cf. 2.5.8.3.1).

2.5.8.5.3 For each variable, a pair of strings may be provided by the client application via the property *Format* (cf. 2.3.2.3.14), which is part of the interface *IRTDVar* (cf. 2.3.1.3.6) which will be used for labelling the respective buttons (an interactive user may alternatively provide a pair of bitmap file names).

2.5.8.5.4 The client application is able to set all properties of this mapped variable via the property *RTDVar* (cf. 2.3.1.3.6) part of the control's interface *IRealtimeDisplay*.

2.5.8.5.5 During runtime, the RTDD will retrieve the initial value for each of the channels as obtained from the RTE and forward these to the control when the control is first displayed.

2.5.8.5.6 Later on, the RTD Multi Switch ActiveX Control will allow an interactive user to set the value of one of the channels mapped to the control's RTD Variables to TRUE which automatically sets the values of all the other channels to FALSE.

2.5.8.5.7 Then, the RTD Multi Switch ActiveX Control will fire the new values for all RTD variables with the event *ChangeValue* (cf. 2.3.1.4.5) to the RTD client.

2.5.8.6 Persistent Data

2.5.8.6.1 This interface initiates that all the RTDVar objects (cf. 2.5.8.3.1) which are mapped to a client variable are stored persistently by their respective interfaces *IRTDVar* (cf. 2.3.2).

2.5.8.6.2 All other properties provided by this interface and also those which can be modified via the property dialogue of the RTD Multi Switch ActiveX Control are stored persistently by this interface.