Proprietary Notice

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

1.1.1 ProDAS is a data acquisition system for gas turbine engine test cells. This specification defines the technical requirements for the interface offered by the Real Time Data Proxy Server, which is used for delivering real time data from the RTE to the clients.

1.1.2 This document defines the COM interface to be used by the clients. Any Windows application, which needs to retrieve real time data from the RTE, should use this interface.

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

ES78031.2739	Real Time Data Provider
DB78026.2787	Real Time Data Proxy Server

1.4 Codes and Standards

N/A

1.5 Abbreviations and Definitions

Capture Rate	The frequency at which the RT Data Server retrieves data from the CVT. In the case of non-buffered data, the capture rate will equal the transfer rate. For buffered data, the capture rate will be set to the lowest common multiple of the scan rates of the channels in a given channel list.
COM	Component Object Model
CVT	Current Value Table
DLL	Dynamic Link Library
ICD	Interface Control Document

IDL	Interface Definition Language
MSDN	Microsoft Developer Network
MTA	Multi-Threaded Apartment
RTDPS	Real Time Data Proxy Server
RTDS	Real Time Data Server
RTE	Real Time Engine
Scan Rate	The rate at which the RTE acquires data.
TCP	Transmission Control Protocol
Transfer rate	The rate of data transfer between the RT Data Proxy Server and the RT Data Server.

2. DESIGN

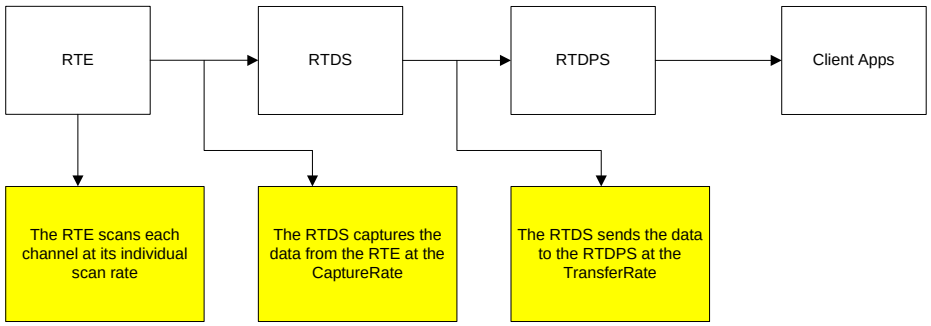
2.1 Introduction

2.1.1 The final product of the RTDPS is a DLL file, which contains an in-process COM component.

2.1.2 The RTDPS uses an apartment-threading model.

2.1.3 The RTDPS is thread-safe.

2.1.4 Diagram



2.2 Interface IRTDProxy

2.2.1 Design

2.2.1.1 Interface: Dual (IDispatch and IUnknown).

2.2.1.2 Automation interface: Yes.

2.2.2 Methods and Properties

2.2.2.1 Property ConnectionString

[propget, id(2), helpstring("property ConnectionString")]
HRESULT ConnectionString([out, retval] BSTR *pVal);

[propput, id(2), helpstring("property ConnectionString")]
HRESULT ConnectionString([in] BSTR newVal);

Argument Name	Description
NewVal	The new connection string to set.
pVal*	Returns the current connection string.

Indicates the information used to establish a connection to a data source. For a socket data source, the connection string shall provide Host, Port/Service and Protocol information, as indicated below:

“Host=<hostname>;Port=< port number>;Protocol=<TCP>”. Or

“Host=<hostname>;Service =< service name >;Protocol=<TCP>”

2.2.2.2 Method CreateChannelList

[id(1), helpstring("method CreateChannelList")]
HRESULT CreateChannelList([out, retval]IChannelList **retVal);

Argument Name	Description
IChannelList**	The new instance of *IChannelList* created by this method. (C.f. 2.3)

To create an instance of an *IChannelList* interface, with no channels specified.

2.2.2.3 Property SecondsToBuffer

[propget, id(3), helpstring("Returns the time period that the interface can buffer, in seconds")]
HRESULT SecondsToBuffer([out, retval] long *pVal);

Argument Name	Description
Pval	Returns the time period that the interface can buffer, in seconds.

Please note that this number is not accurate, the inaccuracy, expressed in seconds, lies below 2/CaptureRate (c.f. 2.3.2.12 for *CaptureRate*).

2.2.3 Events Fired

2.2.3.1 Error

[id(1), helpstring("method Error")]
HRESULT Error([in]long ErrorStatus);

Argument Name	Description
ErrorStatus	The error number indicating the error that just occurred.

Errors occurring during the communication with the data server will cause this event to be fired.

The *Error* event reports a subset of Socket Errors. For more details regarding socket errors, please refer to MSDN.

In addition to the socket errors, the *Error* event also reports some logical errors for debugging purposes. The errors are defined as **enum DS_Error** below:

```
typedef enum DS_Error
{
        DSE_ALREADY_REGISTERED = 1,
        DSE_NOT_REGISTERED,
        DSE_CONNECTION_LOST
}DS_Error;
```

DSE_ALREADY_REGISTERED is fired when the client application tries to:
add a channel to,
remove a channel from,
clear all channels from, or
set the transfer rate to
a channel list, which has already been registered.

DSE_NOT_REGISTERED is fired when the client tries to call the *Refresh*, *CaptureRate* or *GetBufferData* method/property while the corresponding channel list has not been registered.

DSE_CONNECTION_LOST is fired when the connection with the RTDS is lost. Another *Error* event is fired with the socket error number prior to this event.

2.2.3.2 Reconnected

[id(2), helpstring("Event Reconnected")]
HRESULT Reconnected();

This event is fired when the connection to the RTDS is re-established. When the client receives this event, it should unregister all the channel lists and then register them again to commence retrieving channel data. Please note that after re-registering the channel list, the channel list can be different from the previously registered one because technically the configuration of the RTE can change (the RTE restarts with a different configuration) during the period when the connection was not available.

2.2.4 Usage Conditions and Restrictions

2.2.4.1 This interface is used to manage the connection to the data source and to create an instance of *IChannelList*. If an error occurs in communicating with the data source, an *Error* event will be raised.

2.2.4.2 The connection to the data source will be established automatically. This occurs when the *Register* method is invoked from the *IChannelList* associated with the instance of *IChannelList*. The connection will be closed after the last registered (*IsRegister* method returns DS_Registered) *IChannelList* instance invokes the *Unregister* method.

2.2.4.3 If the *ConnectionString* is incorrectly specified, no error will be returned by setting the *ConnectionString* property. However when the *IRTDProxy* interface tries to connect to the data source, an *Error* event will be raised.

2.2.4.4 A client application can create multiple instances of the *IRTDProxy* interface. When multiple instances are created, each instance establishes its own connection to the data source. Different instances can have different *ConnectionString* settings to connect to different data sources. Note that as more instances are created, more resources are used and the network transmission performance reduces.

2.2.5 Persistent Data

2.2.5.1 N/A

2.3 Interface IChannelList

2.3.1 Design

2.3.1.1 Interface: Dual (IDispatch and IUnknown)

2.3.1.2 Automation interface: Yes

2.3.2 Methods and Properties

2.3.2.1 Method Add

[id(1), helpstring("method Add")]
HRESULT Add([in]BSTR ChannelName, [out, retval] IChannel** retVal);

Parameter Name	Description
ChannelName	The name of channel to be added into the list.
IChannel	A reference to the *IChannel* instance just created.

Adds a channel to the *IChannelList* instance. This method can only be called when the *IChannelList* instance is not registered.

If the channel is not configured in the RTE, no error will be returned by this method. However, when the channel list is registered, the *IsConfigured* property (c.f. 2.4.2.3) of any channel which is not configured in the RTE will return DS_Not_Configured, and the *Quality* property (c.f. 2.4.2.6) will return DS_INVALID.

If the channel list is already registered, the *Add* method does nothing and the *retVal is set to NULL; if there is a listener to the *Error* event of the *IRTDProxy* interface, an *Error* event is raised, the *ErrorCode* is set to DSE_ALREADY_REGISTERED and S_OK is returned, otherwise the *Add* method returns DSE_ALREADY_REGISTERED.

2.3.2.2 Method Remove

[id(2), helpstring("method Remove")]
HRESULT Remove([in]VARIANT Key);

Parameter Name	Description
Key	The key word of the channel to be removed from the channel list. Can be a BSTR to indicate the name of the channel to be removed or an integer (i.e. VT_I2 or VT_I4) value indicating the index (0 based) of the channel in the list.

Removes a channel from an *IChannelList* instance. This method can only be called when the *IChannelList* instance is not registered.

If the *Key* indicates a channel that does not exist in the channel list, the *Remove* method does nothing and returns S_OK immediately.

If the channel list is already registered, the *Remove* method does nothing; if there is a listener to the *Error* event of the *IRTDProxy* interface (c.f. 2.2.3.1), an *Error* event is raised with *ErrorCode* set to DSE_ALREADY_REGISTERED and the *Remove* method returns S_OK; otherwise it returns DSE_ALREADY_REGISTERED.

2.3.2.3 Method ClearAll

[id(6), helpstring("method ClearAll")]
HRESULT ClearAll();

Removes all channels from an *IChannelList* instance. This method can only be called when the *IChannelList* instance is not registered.

2.3.2.4 Property _NewEnum

[propget, id(DISPID_NEWENUM), helpstring("property NewEnum")]
HRESULT _NewEnum([out, retval]IUnknown**);

Parameter Name	Description
IUnknown**	The new enumerator interface.

This property if for VB users implementing a For Each … Next loop.

2.3.2.5 Property Item

[propget, id(DISPID_VALUE), helpstring("property Item")]
HRESULT Item([in] VARIANT Key, [out, retval]IChannel** retVal);

Parameter Name	Description
Key	Can be a BSTR to indicate a Channel Name or an integer (i.e. VT_I2 or VT_I4) to indicate the index position (0 based) of the *IChannel* in the *IChannelList*.
IChannel**	The *IChannel* interface to return. (cf. 2.4)

Accesses an *IChannel* instance in *IChannelList*, using the specified channel key. Refer to 2.3.2.2 for more details on the *Key* parameter.

When either the channel name indicated by *Key* does not exist, or the index indicated by *Key* is out of range, *retVal is set to NULL and the property returns S_OK.

2.3.2.6 Property Count

[propget, id(3), helpstring("property Count")]
HRESULT Count([out, retval] unsigned short *pVal);

Parameter Name	Description
long*	The number of channels in the list

Returns the number of channels in *IChannelList*.

2.3.2.7 Method Register

[id(4), helpstring("method Register")]
HRESULT Register([in]BOOL Buffered, [in, optional, defaultvalue(-1.0)] double TransferRate);

Parameter Name	Description
Buffered	To indicate whether to buffer the channels or not.
TransferRate	The TransferRate. In VB, if the TransferRate property has already been used to set the transfer rate, this parameter can be omitted.

Registers the channels in *IChannelList* to the server. When successful, the channels are registered and the channel list editing methods (i.e. *Add*, *Remove* and *ClearAll*) are disabled. Boolean channels can only be defined for an Instantaneous channel list, and not for a Buffered channel list. If Boolean channels are defined for a Buffered channel list, they will be ignored, and treated as if the channels did not exist in the RTE configuration.

2.3.2.8 Method Unregister

[id(5), helpstring("method Unregister")]
HRESULT Unregister();

De-registers the channels previously registered with the server (c.f. 2.3.2.7). Unregistering a channel list will re-enable the channel list editing methods.

2.3.2.9 Method Refresh

[id(7), helpstring("method Refresh")]
HRESULT Refresh([out, retval]BOOL *retVal);

Parameter Name	Description
BOOL*	Return value to indicate that the channel list is updated with a newer data frame.

Notifies the *IChannelList* to scroll to the next frame of buffered data. For a buffered channel list, it returns TRUE when there is a buffered frame of data, and FALSE when there are no newer data. For instantaneous channels, it always returns TRUE, which means for the instantaneous channels, the most recently received data will always be returned to the client application. If the corresponding channel list is not registered, *Refresh* always returns FALSE.

2.3.2.10 Method GetBufferedData

[id(11), helpstring("method GetBufferedData")]
HRESULT GetBufferedData([out]long *pNum, [out, size_is(*pNum)] ChannelData **pBuffer) ;

Parameter Name	Description
*pNum	A pointer pointing to a long variable. On return, the value of the long variable will be the number of frames of channel data in the *pBuffer* array.
pBuffer	A reference of a pointer. The pointer will carry back the address of an array of the *ChannelData* data type (c.f. 2.5.1.2). The *ChannelData* array is defined as follows, in C/C++ code: ChannelData pBuffer[*pNum] Note that the client application must release this buffer.

The *GetBufferedData* method provides a way to retrieve channel data in a batch mode. This method cannot be used with the *IChannel* interface properties or the *Refresh* method. Using the *GetBufferedData* method and the *Refresh* method or the *IChannel* interface in the same *IChannelList* instance can cause problems.

The *GetBufferedData* method is for buffered channels only. Do not call this method for instantaneous channel lists.

The channel data are stored in the buffer in the following order:

$Data_{CH0T0}$, $Data_{CH1T0}$, … $Data_{CHnT0}$, $Data_{CH0T1}$, $Data_{CH1T1}$, … $Data_{CHnT1}$, $Data_{CH0Tm}$, $Data_{CH1Tm}$, … $Data_{CHnTm}$

If the client application wants to access the 6th channel in the 3rd time frame, it can use the statement:

ChannelData *pCh = pBuffer[2 * numofchannel + 5];

where *numofchannel* defines the number of channels in the list.

For example, suppose that the following two channels are in a channel list:

Name	Scanrate	Datatype
CH1	50HZ	FLOAT
CH2	20HZ	FLOAT

At any given moment, if the client application calls *GetBufferedData*, it may retrieve a data buffer containing the following data:

Time frame	CH1		CH2	
	VALUE	QUALITY	VALUE	QUALITY
0ms	1.0	DS_GOOD	2.0	DS_GOOD
10ms	1.0	DS_REPEAT	2.0	DS_REPEAT
20ms	1.1	DS_GOOD	2.0	DS_REPEAT
30ms	1.1	DS_REPEAT	2.0	DS_REPEAT
40ms	1.2	DS_BAD	2.0	DS_REPEAT
50ms	1.2	DS_REPEAT	2.2	DS_GOOD
60ms	1.1	DS_GOOD	2.2	DS_REPEAT
70ms	1.1	DS_REPEAT	2.2	DS_REPEAT
80ms	1.0	DS_GOOD	2.2	DS_REPEAT
90ms	1.0	DS_REPEAT	2.2	DS_REPEAT
…				

Please note that the data buffer can start from any time frame (i.e. 0ms, 10ms, 20ms, 30ms, 40ms…).

After calling the *GetBufferedData* method, the buffer will be empty.

2.3.2.11 Property TransferRate

[propget, id(8), helpstring("property TransferRate ")]
HRESULT TransferRate([out, retval] double *pVal);

[propput, id(8), helpstring("property TransferRate ")]
HRESULT TransferRate([in] double newVal);

Parameter Name	Description
double *	Returns the current transfer rate setting in Hz.
newValue	The new transfer rate to set

The *TransferRate* property indicates how frequently the channel data are transmitted from the RTDS. Any value between 0.01 Hz and 10 Hz is acceptable, however, the maximum value of 10 Hz can be limited by the maximum rate defined for the RTDS and by the scan rate of the RTE. The RTDS will specify the closest number that is supported by the RTE when the *IChannelList* is registered. When the requested *TransferRate* is less than 0.01 Hz, the RTDS will set it to 1 Hz, and when it is greater than the maximum (nominally 10 Hz), the RTDS will set it to its maximum transfer rate.

If the *ChannelList* is not registered, the *TransferRate* property returns the value previously set by the client application.

When the channel list is registered, the *TransferRate* property returns the actual transfer rate that the RTE supports and the value of this property is not allowed to be changed.

2.3.2.12 Property CaptureRate

[propget, id(10), helpstring("property CaptureRate")]
HRESULT CaptureRate([out, retval] double *pVal);

Parameter Name	Description
double **	Returns the current capture rate setting in Hz.

The *CaptureRate* is designed for buffered channels. It indicates how frequently the RTDS will capture the real time data. For an instantaneous channel list, it always returns the same value as the TransferRate.

If the channel list is not registered, the *CaptureRate* will return 0.0.

2.3.2.13 Method IsRegistered

[id(9), helpstring("method IsRegistered")]
HRESULT IsRegistered([out, retval] DS_RegisteredStatus *retVal);

Parameter Name	Description
DS_RegisteredStatus*	Returns the currently registered status. DS_Registered indicates that the channel list is registered. DS_Not_Registered indicates that the channel list is not yet registered. DS_No_Configured_Channel indicates that the channel list does not contain any channel which is configured by the RTE; please note that this status is NOT considered as successfully registered, the behaviour is the same as for a channel list that is in the state of DS_Not_Registered.

Use this method to indicate whether or not the channel list is registered. Note, even if there are some channels that are invalid, the *IsRegistered* method still returns DS_Registered if at least one channel is successfully registered. In this case, when accessing the invalid channels through the *IChannel* interface, the *IsConfigured* property returns FALSE.

2.3.3 Events Fired

There are no events.

2.3.4 Usage Conditions and Restrictions

2.3.4.1 The *IChannelList* interface cannot be created directly. An instance of *IChannelList* can be retrieved by calling the *CreateChannelList* method of the *IRTDProxy* interface (cf. 2.2.2.2).

2.3.4.2 Before calling the *Register* method, all channels should be added into the *IChannelList* by calling the *Add* method. At least one channel should be added or the *Register* method will fail. Client applications should not call the *Add*, *Remove* or *ClearAll* methods after successfully calling the *Register* method.

2.3.4.3 For the channel lists which have been registered successfully (*IsRegister* returns DS_Registered, c.f. 2.3.2.13) using the *Register* method (c.f. 2.3.2.7), the *Unregister* (c.f. 2.3.2.8) method should be called when the client application has finished using the *IChannelList* interface.

2.3.4.4 Although the *IChannel* object is still valid, once the *Remove* or *ClearAll* method is called, the *IChannel* object(s) removed from the *IChannelList* is (are) no longer valid. Accessing the properties of an *IChannel*, which has been removed from the *IChannelList*, will cause unpredictable results.

2.3.4.5 The dynamic channel data in the *IChannel* object will not refresh until a *Refresh* (cf. 2.3.2.9) call is invoked, even if a newer frame of data is already in the buffer.

2.3.4.6 Zero (0) Hz channels can be defined in both a Buffered channel list and an Instantaneous channel list. In both cases, the value of these channels will be updated at the transfer rate.

2.3.4.7 Boolean channels can only be defined for an Instantaneous channel list. If Boolean channels are defined for a Buffered channel list, they will be ignored, and treated as if the channels did not exist in the RTE configuration.

2.3.5 Persistent Data

2.3.5.1 N/A

2.4 Interface IChannel

2.4.1 Design

2.4.1.1 Interface: Dual (IDispatch and IUnknown)

2.4.1.2 Automation interface: Yes

2.4.1.3 This interface is intended to provide a user-friendly way to access individual channel attributes.

2.4.2 Methods and Properties

2.4.2.1 Property Name

[propget, id(12), helpstring("property channel name")]
HRESULT Name([out, retval] BSTR *pVal);

[propput, id(12), helpstring("property channel name")]
HRESULT Name([in] BSTR newVal);

Parameter Name	Description
BSTR *	Channel name to retrieve.
NewVal	New channel name to set.

2.4.2.2 Property Type

[propget, id(13), helpstring("property Channel Type")]
HRESULT Type([out, retval] BSTR *pVal);

Parameter Name	Description
pVal	Returns the channel type. “FLOAT” indicates the channel is a float channel; “DISCRETE” indicates the channel is a Boolean channel; “ERROR” indicates that an error occured.

2.4.2.3 Property IsConfigured

[propget, id(15), helpstring("property IsConfigured")]
HRESULT IsConfigured([out, retval] DS_ConfiguredEnum *pVal);

Parameter Name	Description
PVal	Returns the channel configured status. DS_Not_Configured indicates that the channel is not configured. DS_Configured indicates that the channel is configured. DS_Unknown indicates that the channel list is not registered.

2.4.2.4 Property Value

[propget, id(0), helpstring("property Value")]
HRESULT Value([out, retval] VARIANT *pVal);

Parameter Name	Description
pVal	The current channel value (EU converted if applicable). For a float channel, returns a double value. For a Boolean channel, returns a BOOL value (1 for TRUE, 0 for FALSE). The quality will indicate if the value is not valid.

2.4.2.5 Property RawValue

[propget, id(1), helpstring("property RawValue")]
HRESULT RawValue([out, retval] double *pVal);

Parameter Name	Description
pVal	Raw value of the channel before EU conversion (if applicable). Applicable for some float channels only. The quality will indicate when the value is not valid.

2.4.2.6 Property Quality

[propget, id(2), helpstring("property Quality")]
HRESULT Quality([out, retval] DS_Quality *pVal);

Parameter Name	Description
pVal	Returns one of the following values: DS_INVALID, DS_GOOD, DS_SUSPECT, DS_BAD, or DS_REPEAT, indicating initial value, good quality, suspect quality, bad quality or a repeated value, respectively. The DS_INVALID quality indicates that the channel has not been registered or it has been registered but the data from the RTE has not been received.

2.4.2.7 Property ValueLF

[propget, id(3), helpstring("property Value from last fullset")]
HRESULT ValueLF([out, retval] double *pVal);

Parameter Name	Description
pVal	Channel value from the last fullset. The *QualityLF* property (cf. 2.4.2.8) indicates the quality of this value.

2.4.2.8 Property QualityLF

[propget, id(4), helpstring("property quality from last fullset")]
HRESULT QualityLF([out, retval]DS_Quality *pVal);

Parameter Name	Description
pVal	Channel quality from the last fullset. Returns one of the following values: DS_GOOD indicating good quality, DS_SUSPECT indicating suspect quality, or DS_BAD indicating bad quality or not applicable (i.e. channel not registered, Error occurred).

2.4.2.9 Property QualityCeiling

[propget, id(5), helpstring("property QualityCeiling")]
HRESULT QualityCeiling([out, retval] DS_Quality *pVal);

Parameter Name	Description
pVal	Channel quality ceiling. Returns one of the following values: DS_GOOD, DS_SUSPECT or DS_BAD indicating good quality, suspect quality, and bad quality respectively.

2.4.2.10 Property Alarm

[propget, id(6), helpstring("property Alarm")]
HRESULT Alarm([out, retval] DS_AlarmStatus *pVal);

Parameter Name	Description	
pVal	Channel alarm status. Returns one of the following strings:	
	OK	Status is OK
	LO	Status is LO, for float channels only
	HI	Status is HI, for float channels only
	LOLO	Status is LOLO, for float channels only
	HIHI	Status is HIHI, for float channels only
	ROC	Rate of Change, for float channels only
	BOOLEAN_ALARM	Channel is in an alarm state, for Boolean channels only.
	BAD_VALUE	This value indicates that this field cannot be used. The possible reasons could be either the corresponding channel is not configured with an Alarm, or real time data has not been correctly received from the RTE.

2.4.3 Property LOLO

[propget, id(10), helpstring("property LOLO")]
HRESULT LOLO([out, retval] double *pVal);

Parameter Name	Description
pVal	Returns the LOLO limit value for a float channel. A value of -99999.0 indicates that an error occurred or this property is not applicable (Boolean channel).

2.4.4 Property LO

[propget, id(9), helpstring("property LO")]
HRESULT LO([out, retval] double *pVal);

Parameter Name	Description
pVal	Returns the LO limit value for a float channel. A value of -99999.0 indicates that an error occurred or this property is not applicable (Boolean channel).

2.4.5 Property HI

[propget, id(8), helpstring("property HI")]
HRESULT HI([out, retval] double *pVal);

Parameter Name	Description
double*	Returns the HI limit value for a float channel. A value of -99999.0 indicates that an error occurred or this property is not applicable (Boolean channel).

2.4.6 Property HIHI

[propget, id(7), helpstring("property HIHI")]
HRESULT HIHI([out, retval] double *pVal);

Parameter Name	Description
double*	Returns the HIHI limit value for a float channel. A value of -99999.0 indicates that an error occurred or this property is not applicable (Boolean channel).

2.4.7 Property RedirectChannel

[propget, id(14), helpstring("property RedirectChannel")]
HRESULT RedirectChannel([out, retval] long *pVal);

Parameter Name	Description
pVal*	If the current channel's value is not redirected to any channel, this value returns 0, if the channel's value is redirected to a constant value, this value returns –1, otherwise this value returns the channel ID from which the current channel obtains its channel value.

2.4.8 Events Fired

There are no events.

2.4.9 Usage Conditions and Restrictions

2.4.9.1 The *IChannel* interface cannot be instantiated directly. An instance of the *IChannel* interface can be created by calling the *Add* method of the *IChannelList* interface (cf. 2.3.2.1)

2.4.9.2 Properties are accessible only when the corresponding *IChannelList* instance is registered (a successful *Register* method is invoked).

2.4.10 Persistent Data

N/A

2.5 Structure ChannelData

2.5.1 Design

2.5.1.1 ChannelData is defined for the client to access the buffered channel data in batch mode (c.f. 2.3.2.10).

2.5.1.2 IDL

```
typedef struct tagChannelData
{
        double          dblValue;
        DS_Quality      Quality;
        DS_AlarmStatus  AlarmStatus;
}ChannelData;
```

2.6 Examples

2.6.1 Visual C++ example

```
#import "rtdps.dll"
#include <stdio.h>

int main()
{
	RTDPSLib::IChannelPtr theChannel, aChannel;
	RTDPSLib::IChannelListPtr theList;
	RTDPSLib::IRTDProxyPtr theProxy;
	_variant_t vCount, vTemp;
	BOOL bMore = TRUE;

	CoInitialize(NULL);
	try
	{
		HRESULT hr =
	theProxy.CreateInstance(__uuidof(RTDPSLib::RTDProxy));

		theList = theProxy->CreateChannelList();

		theChannel = theList->Add(L"GoodChannelName");
		//do something with theChannel if you want
		//And then release it.
		theChannel.Release();

		//Register at 10hz
		theList->Register(FALSE, 10.0);
	}catch(…)
	{
		//Have your own exception handling code here.
		return –1;
	}

	while(bMore)
	{
		try
		{
			for(int i = 0; i < theList->GetCount(); i ++)
			{
				vCount = (short)i;
				aChannel = theList->GetItem(vCount);
```

```
                        //do something with the value of aChannel
                        vTemp = aChannel->GetValue();
                        //If the channel is a Boolean channel, use intVal
                        printf("%d\n", vTemp.intVal);

                        //Release it after used
                        aChannel.Release();
                }
                //Make sure to call Refresh and wait for TRUE come back
                //When theList is registered as instantaneous channel list,
                // theList->Refresh() always returns true.
                while(bMore && !theList->Refresh());
            }catch(...)
            {
                //if anything wrong, quit
                bMore = FALSE;
            }
        }

        theList.Unregister();
        theList.Release();
        theProxy.Release();
        CoUninitialize();
        return 0;
}
```

2.6.2 VB example

```
Public Sub main()
        Dim oProxy as new RTDProxy
        Dim oChList as ChannelList
        Dim oCh as Channel
        Dim bMore as Boolean

        oProxy.ConnectionString = “Host=picard;Port=50001;Protocol=TCP”
        Set oChList = oProxy.CreateChannelList()
        Set oCh = oChList.Add(“ChannelName”)
        ‘Do something with the new created channel instance and then release it.
        ‘Repeat calling Add method to add more channels to the Channel list.
        Set oCh = nothing
        ‘Register buffered channel at 10 hz
        oChList.Register(True, 10.0)

        bMore = True

        On Error Goto ErrHandle
        While bMore
                For each oCh in oChList
                        Debug.Print oCh.Value
                Next
                While bMore And (Not oChList.Refresh())
                Wend
        Wend
        Exit Sub

ErrHandle:
        bMore = False
        Resume Next
End Sub
```