Proprietary Notice

This document is the property of MDS Aero Support Corporation, and is provided on condition that it be used exclusively for evaluation purposes. Any duplication or reproduction, in whole or in part, without prior written consent of an authorized MDS Aero Support Corporation representative is prohibited.

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 Trace Utility, which is used for logging informational messages to a trace file.

1.1.2 This document defines the COM interface to be used by the clients. Any Windows application in proDAS should use this interface when logging informational or debug messages. All clients of the Trace Utility on the same computer should use the same trace file for logging purposes, so that all messages can be viewed in the same file.

1.1.3 The trace information written to the trace file will consist of four individual parts:

Title	**Description**
Date and Time	Automatically added by the Trace Utility. The date and time will be in the format: DD-MMM-YYYY HH:MM:SS
Module ID	A global module identifier, indicating the module that is generating the trace message. The allocation of these module identifiers needs to be co-ordinated between all application developers.
Source	An identifier of the method (function) within the module that is generating the trace message.
Description	The detailed trace message.

1.1.4 The Trace Utility provides an automatic backup feature for the trace file.

1.1.4.1 When the trace file grows bigger than a certain size (cf. 2.1.6.7), the Trace Utility will backup the current trace file into a specified directory (cf. 2.1.6.6).

1.1.4.2 The backup destination file name is SYSTEM<yyyymmddhhiiss>.txt, comprised of an upper case SYSTEM followed by a 4 digit year, 2 digit month, 2 digit day, 2 digit hour, 2 digit minute and 2 digit second with a .txt extension.

1.1.4.3 The backup feature will be disabled if either the initialisation file is missing or the necessary information in the initialisation file is incorrect. Details of the initialisation file and information are described in section 2.1.6.

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

ES78001.2620 Functional Requirements Document for ProDAS

1.4 Codes and Standards

N/A

1.5 Abbreviations and Definitions

API	Application Programming Interface
COM	Component Object Model
DLL	Dynamic Link Library
ICD	Interface Control Document
IDL	Interface Definition Language
proDAS	Professional Data Acquisition System
STA	Single-Threaded Apartment
UML	Unified Modelling Language

2. DESIGN

2.1 Introduction

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

2.1.2 In addition, this DLL also provides API calls for situations when a COM interface is not convenient to use.

2.1.3 The Trace Utility uses a single threaded apartment model (STA).

2.1.4 The Trace Utility is process-safe, i.e. it will function correctly when accessed simultaneously by two or more applications.

2.1.5 The Trace Utility is thread-safe.

2.1.6 Initialisation File

2.1.6.1 The Trace Utility uses an initialisation file to store its configuration information.

2.1.6.2 The initialisation file is named MDSComm.ini, and will be located in the <installed directory>\..\Ini\ directory, where the <installed directory> is the directory in which the Trace Utility (MDSComm.dll) is installed and registered.

2.1.6.3 When registering the component, the Trace Utility checks the <installed directory>\..\Ini directory for the MDSComm.ini file. If this file does not exist, the Trace Utility will create a new one automatically with a [Trace] section containing keys for the “DefaultTraceFile”, “BackupPath”, “MaxLogFileSize” and “Verbosity” with default values (cf. 2.1.6.5, 2.1.6.6, 2.1.6.7, 2.1.6.8).

2.1.6.4 If such a file (MDSComm.ini) already exists but doesn’t contain either the [Trace] section or the “DefaultTraceFile”, “BackupPath”, “MaxLogFileSize” or “Verbosity” keys under the [Trace] section, the Trace Utility will setup these keys with their default values (cf. 2.1.6.5, 2.1.6.6, 2.1.6.7, 2.1.6.8).

2.1.6.5 The Trace Utility will set “C:\TEMP\SYSTEM.TXT” as the default trace file name. If the C:\TEMP directory does not exist, the Trace Utility will create it.

2.1.6.6 The Trace Utility will search the %temp%, %tmp% and then “Temp” directory in the root directory on each logical hard drive, and use the first existing one as the default value of the “BackupPath” key. If none of these directories exist, the key will be left empty, which means the Backup feature (cf. 1.1.4) will be disabled. In this case, the Trace Utility will display a message box to notify the operator (cf. Figure 1).

2.1.6.7 The Trace Utility will always set the default value of “MaxLogFileSize” to 524288 (0x80000).

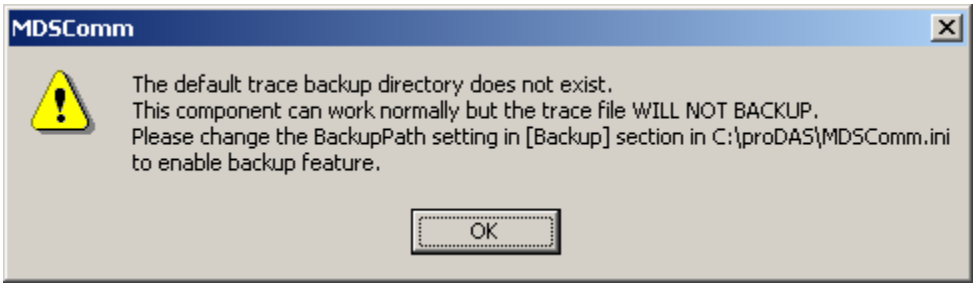

Figure 1 Backup path not found

Please note the path in the Figure1 (C:\proDAS”) is based on the install directory.

2.1.6.8 The Trace Utility will always set the default value of “Verbosity” to 5 (vbLowest).

2.2 Interface ITrace

2.2.1 Design

2.2.1.1 This interface is an automation interface.

2.2.2 Methods and properties

2.2.2.1 Method Open

2.2.2.1.1 This method is used for backward compatibility; new applications should use the *TraceFileName* property instead.

[id(1), helpstring("method Open")] HRESULT Open([in]BSTR FileName);

Argument Name	Description
FileName	Name of the trace file to which all trace information will be written.

2.2.2.1.2 This method initialises internal logic to prepare for any further action. If the trace file does not exist, a new file will be created the first time that a *Write*, *WriteArgs*, *WriteEx*, or *WriteArgsEx* method is called. If the file already exists, trace information will be appended to the end of the existing file.

2.2.2.1.3 If the file indicated by the input parameter is read-only, or the interactive user doesn't have enough rights to access the file, an error will occur at the first call to the *Write*, *WriteArgs*, *WriteEx*, or *WriteArgsEx* method.

2.2.2.1.4 No error is generated in this call.

2.2.2.2 Method Write

[id(2), helpstring("method Write")] HRESULT Write([in]BSTR ModuleID, [in]BSTR Source, [in]BSTR Description, [in, optional]VARIANT Arg1, [in, optional]VARIANT Arg2, [in, optional]VARIANT Arg3, [in, optional]VARIANT Arg4, [in, optional]VARIANT Arg5, [in, optional]VARIANT Arg6, [in, optional]VARIANT Arg7, [in, optional]VARIANT Arg8);

Argument Name	Description
ModuleID	A maximum 3-character string that identifies the module generating the trace message. This *ModuleID* should be a unique name to differentiate between modules. All characters after the third one will be discarded.
Source	A maximum 28-character string that identifies the function (method) generating the trace message. All characters after the 28th one will be discarded.
Description	A string containing detailed trace information. The C printf format string is allowed. The optional Arg1 … Arg8 arguments provide the additional parameters for the format string. These parameters must match the format string.
Arg1 – Arg8	The additional parameters for the C printf format string indicated by *Description*.

2.2.2.2.1 This method writes trace information with up to 8 arguments into the trace file indicated by the *Open* method or the *TraceFileName* property.

2.2.2.2.2 This method provides a convenient interface for C++ code. If called from C code, the best approach is to use the sprintf family of functions to generate the *Description*, and omit all *Argn* parameters. For a VB user, the most flexible way is to use the *WriteArgs* method.

2.2.2.2.3 It is not recommended to include the cr (0x0d) and lf (0x0a) characters as parameters to force a line break in the trace information.

2.2.2.2.4 The system error code will be returned as the HRESULT code if an error occurs during the write operation.

2.2.2.3 Method WriteArgs

[id(3), vararg, helpstring("method WriteArgs")] HRESULT WriteArgs([in]BSTR ModuleID, [in]BSTR Source, [in]BSTR Description, [vararg, in]SAFEARRAY(VARIANT)* Args);

Argument Name	Description
ModuleID	A maximum 3-character string that identifies the module generating the trace message. This *ModuleID* should be a unique name to differentiate between modules. All characters after the third one will be discarded.
Source	A maximum 28-character string that identifies the function (method) generating the trace message. All characters after the 28th one will be discarded.
Description	A string containing detailed trace information. The C printf format string is allowed. The optional arguments provide the additional parameters for the format string. These parameters must match the format string.
Args	The additional parameters for the C printf format string indicated by *Description*.

2.2.2.3.1 This method writes trace information with unlimited additional arguments into the trace file indicated by the *Open* method or the *TraceFileName* property.

2.2.2.3.2 This method provides a convenient interface for C++ code. If called from C code, the best approach is to use the sprintf family of functions to generate the *Description*, and omit all additional parameters.

2.2.2.3.3 It is not recommended to include the cr (0x0d) and lf (0x0a) characters as parameters to force a line break in the trace information.

2.2.2.3.4 The system error code will be returned as the HRESULT code if an error occurs during the write operation.

2.2.2.4 Method WriteEx

[id(6), helpstring("method WriteEx")] HRESULT WriteEx([in, defaultvalue(vbLowest)]VerbosityEnum TraceLevel, [in]BSTR ModuleID, [in]BSTR Source, [in]BSTR Description, [in, optional]VARIANT Arg1, [in, optional]VARIANT Arg2, [in, optional]VARIANT Arg3, [in, optional]VARIANT Arg4, [in, optional]VARIANT Arg5, [in, optional]VARIANT Arg6, [in, optional]VARIANT Arg7, [in, optional]VARIANT Arg8);

Argument Name	**Description**
TraceLevel	VerbosityEnum (cf. Table 2-1 VerbosityEnum), specifies a 1 to 5 verbosity level. Only the messages with a *TraceLevel* greater than or equal to *Verbosity* (cf. 2.2.2.7) will be written into the trace file.
ModuleID	A maximum 3-character string that identifies the module generating the trace message. This *ModuleID* should be a unique name to differentiate between modules. All characters after the third one will be discarded.
Source	A maximum 28-character string that identifies the function (method) generating the trace message. All characters after the 28th one will be discarded.
Description	A string containing detailed trace information. The C printf format string is allowed. The optional Arg1 … Arg8 arguments provide the additional parameters for the format string. These parameters must match the format string.
Arg1 – Arg8	The additional parameters for the C printf format string indicated by *Description*.

2.2.2.4.1 This method writes trace information with up to 8 arguments into the trace file indicated by the *Open* method or the *TraceFileName* property.

2.2.2.4.2 This method provides a convenient interface for C++ code. If called from C code, the best approach is to use the sprintf family of functions to generate the *Description*, and omit all *Argn* parameters. For a VB user, the most flexible way is to use the *WriteArgsEx* method.

2.2.2.4.3 It is not recommended to include the cr (0x0d) and lf (0x0a) characters as parameters to force a line break in the trace information.

2.2.2.4.4 The system error code will be returned as the HRESULT code if an error occurs during the write operation.

2.2.2.5 Method WriteArgsEx

[id(7), vararg, helpstring("method WriteArgsEx")] HRESULT WriteArgsEx([in, defaultvalue(vbLowest)]VerbosityEnum TraceLevel, [in]BSTR ModuleID, [in]BSTR Source, [in]BSTR Description, [in]SAFEARRAY(VARIANT)* Args);

Argument Name	**Description**
TraceLevel	VerbosityEnum (cf. Table 2-1 VerbosityEnum), specifies a 1 to 5 verbosity level. Only the messages with a *TraceLevel* greater than or equal to *Verbosity* (cf. 2.2.2.7) will be written into the trace file.
ModuleID	A maximum 3-character string that identifies the module generating the trace message. This *ModuleID* should be a unique name to differentiate between modules. All characters after the third one will be discarded.
Source	A maximum 28-character string that identifies the function (method) generating the trace message. All characters after the 28^{th} one will be discarded.
Description	A string containing detailed trace information. The C printf format string is allowed. The optional arguments provide the additional parameters for the format string. These parameters must match the format string.
Args	The additional parameters for the C printf format string indicated by *Description*.

2.2.2.5.1 This method writes trace information with unlimited additional arguments into the trace file indicated by the *Open* method or the *TraceFileName* property.

2.2.2.5.2 This method provides a convenient interface for C++ code. If called from C code, the best approach is to use the sprintf family of functions to generate the *Description*, and omit all additional parameters.

2.2.2.5.3 It is not recommended to include the cr (0x0d) and lf (0x0a) characters as parameters to force a line break in the trace information.

2.2.2.5.4 The system error code will be returned as the HRESULT code if an error occurs during the write operation.

2.2.2.6 Property TraceFileName

[propget, id(4), helpstring("property TraceFileName")] HRESULT TraceFileName([out, retval] BSTR *pVal);

[propput, id(4), helpstring("property TraceFileName")] HRESULT TraceFileName([in] BSTR NewName);

Argument Name	Description
BSTR*	Returns the file name that was set when calling the *Open* method, or the default trace file name otherwise.
NewName	New trace file name to use to write all trace messages.

2.2.2.6.1 Get and set the trace file name. If the trace file name is not set explicitly, the Trace Utility will use the default one defined in the MDSComm.ini file.

2.2.2.7 Property Verbosity

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

[propput, id(5), helpstring("property Verbosity")] HRESULT Verbosity([in] VerbosityEnum newVal);

Argument Name	Description
VerbosityEnum*	Returns the current verbosity setting. (cf. Table 2-1 VerbosityEnum),
NewVal	VerbosityEnum, (cf. Table 2-1 VerbosityEnum). Specifies the new verbosity level.

2.2.2.7.1 The *Verbosity* property indicates one of five different reporting levels. When the *WriteArgsEx* (cf. 2.2.2.5) or the *WriteEx* (cf. 2.2.2.4) methods are called, the Trace Utility will compare the *TraceLevel* parameter against this value. If the *TraceLevel* is greater than or equal to the *Verbosity* level, the corresponding message will be written into the trace file, otherwise, it will be ignored.

2.2.2.7.2 The default value of *Verbosity* is 5 (vbLowest), which can be changed by modifying the MDSCOMM.ini file.

2.2.2.7.3 The following is the definition of VerbosityEnum:

```
typedef enum VerbosityEnum
{
        vbHighest = 1,
        vbAboveNormal,
        vbNormal,
        vbBelowNormal,
        vbLowest
}VerbosityEnum;
```

Table 2-1 VerbosityEnum

2.2.2.8 Property WaitTimeout

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

[propput, id(8), helpstring("property WaitTimeout")] HRESULT WaitTimeout([in] long newVal);

Argument Name	Description
long*	Returns the current wait timeout setting in milliseconds; the default value is 100.
NewVal	Specifies the new wait timeout value.

2.2.2.8.1 *WaitTimeout* indicates the maximum amount of time that the Trace Utility will wait for the trace file to become available. The Trace Utility is operating in a multiple process and multiple thread environment. When one process/thread is trying to write a trace message to the same trace file, the Trace Utility will wait for the other process/thread to finish the tracing operation. If the Trace Utility cannot get the control of the trace file after the *WaitTimeout* (in millisecond) period, the Trace Utility drops the trace message and returns without reporting an error.

2.2.2.8.2 Any time critical trace message writer can set the *WaitTimeout* value small enough to avoid a long delay for tracing.

2.2.2.8.3 On the other hand, any non-time critical trace message writer can increase the *WaitTimeout* value to solve the issue of missing trace messages; however doing so can affect the response time of the application and increase the possibility of lost trace messages for the other trace writers writing to the same trace file.

2.2.2.8.4 Please refer to 2.2.4.5 for detailed information on how to reduce the number of missing trace messages.

2.2.3 Events Fired

There are no events.

2.2.4 Usage Conditions and Restrictions

2.2.4.1 All clients of the Trace Utility on the same computer should use the same trace file for logging purposes, so that all messages can be viewed in the same file.

2.2.4.2 The Trace Utility can prevent multi-access file conflicts from occurring when multiple processes access the same trace file, however in the following cases a file access conflict (write error) may still happen:

- The trace file is opened for writing by a program other than through the Trace Utility.
- The trace file is a shared network file, and is accessed from multiple computers through the Trace Utility.
- By using a DOS tool like *subst* (substitute) to map file systems so that more than one absolute path exists, and different absolute paths are used in different accesses of the same trace file.

2.2.4.3 The Trace Utility will trap all errors associated to these file access conflicts.

2.2.4.4 It is recommended to use the *WriteEx* (cf. 2.2.2.4) and *WriteArgsEx* (cf. 2.2.2.5) methods instead of *Write* (cf. 2.2.2.2) and *WriteArgs* (cf. 2.2.2.3). *WriteEx* and *WriteArgsEx* provide the capability of specifying the *TraceLevel* while *Write* and *WriteArgs* internally call *WriteEx* and *WriteArgsEx* with the *TraceLevel* hard coded to vbLowest (5).

2.2.4.5 Missing trace messages:

2.2.4.5.1 In case many tracing requests are sent to the same trace file at the same time (or within a very short period), the Trace Utility can drop some of the requests.

2.2.4.5.2 To avoid or reduce missing trace messages, the client application can try the following solutions in order:

- Reduce the quantity of messages being traced;
- Utilise the *TraceLevel* parameter to group the trace messages into different severity levels, to minimise the number of critical trace messages being logged;
- Combine multiple messages into one wherever possible;
- Write the trace messages to a unique trace file; and
- Increase the timeout by setting the *WaitTimeout* property (c.f. 2.2.2.8) if timing is not critical for the client application.

2.2.4.5.3 The Trace Utility reports NO ERRORS, even when the trace message is dropped.

2.2.5 Persistent Data

The MDSComm.ini initialisation file is used to store configuration information (cf. 2.1.6).

2.2.6 Examples

2.2.6.1 VB Example

```
Public Sub main()
   Dim o As Object
   Set o = CreateObject("MDSCOMM.TRACE")
   o.TraceFileName = "C:\test.log"
   o.Write "PJ1", "Sub main()", "%s", o.TraceFileName
   o.WriteArgs "PJ1", "Sub main()", "Trace file name:%s  %d", o.TraceFileName, Hour(Now)
End Sub
```

This sample code will generate the following messages in the trace file:

```
15-Apr-2002 09:52:48    PJ1    Sub main()    c:\test.log
15-Apr-2002 09:52:49    PJ1    Sub main()    Trace file name:C:\test.log  9
```

2.2.6.2 VC++ (smart pointer) Example

```
#import "..\Debug\mdscomm.dll" no_namespace
int main()
{
	ITracePtr pTrace;
	CoInitialize(NULL);
	HRESULT hr = pTrace.CreateInstance(__uuidof(Trace));
	if(FAILED(hr))
		return -1;

	pTrace->TraceFileName = L"c:\\test.log";
	pTrace->Write(L"TST", L"main()", L"Nothing important %d", 1l);
	return 0;
}
```

This sample code will generate the following messages in the trace file:

20-Sep-2002 11:36:58 TST main() Nothing important 1

2.2.6.3 VC++ (smart pointer) Example for Verbosity usage

```
#import "MDSCOMM.DLL" no_namespace
#include <iostream>
#include <assert.h>
#include "mdscommdll.h"
using namespace std;

int main()
{
	HRESULT hr = CoInitialize(NULL);
	ITracePtr trace;
	long n = 1;

	assert(hr == S_OK);

	hr = trace.CreateInstance(__uuidof(Trace));
	assert(hr == S_OK);

	cout<<"Current Trace file : "<<(LPCTSTR)trace->TraceFileName<<endl;
	cout<<"Testing COM Interface\n\n"<<endl;
	cout<<"Current verbosity setting: "<<trace->Verbosity<<endl;

	trace->WriteEx(vbLowest, "COM", "Source", "Descrption %d", n ++);
	trace->WriteEx(vbBelowNormal, "COM", "Source", "Descrption %d", n ++);
	trace->WriteEx(vbNormal, "COM", "Source", "Descrption %d", n ++);
	trace->WriteEx(vbAboveNormal, "COM", "Source", "Descrption %d", n ++);
```

```
        trace->WriteEx(vbHighest, "COM", "Source", "Descrption %d", n ++);

        cout<<"Now set the verbosity to vbBelowNormal"<<endl;
        trace->Verbosity = vbBelowNormal;
        cout<<"Current verbosity setting: "<<trace->Verbosity<<endl;
        cout<<"The "<<n <<" line should not be written into trace file"<<endl;
        trace->WriteEx(vbLowest, "COM", "Source", "Descrption %d", n ++);
        trace->WriteEx(vbBelowNormal, "COM", "Source", "Descrption %d", n ++);
        trace->WriteEx(vbNormal, "COM", "Source", "Descrption %d", n ++);
        trace->WriteEx(vbAboveNormal, "COM", "Source", "Descrption %d", n ++);
        trace->WriteEx(vbHighest, "COM", "Source", "Descrption %d", n ++);

        trace.Release();
        CoUninitialize();
        return 0;
}
```

The following trace message are written into trace file

14-Jan-2003 09:36:34	COM	Source	Descrption 1
14-Jan-2003 09:36:34	COM	Source	Descrption 2
14-Jan-2003 09:36:34	COM	Source	Descrption 3
14-Jan-2003 09:36:34	COM	Source	Descrption 4
14-Jan-2003 09:36:34	COM	Source	Descrption 5
14-Jan-2003 09:36:34	COM	Source	Descrption 7
14-Jan-2003 09:36:34	COM	Source	Descrption 8
14-Jan-2003 09:36:34	COM	Source	Descrption 9
14-Jan-2003 09:36:34	COM	Source	Descrption 10