Voltech
http://www.voltech.com

Technical note 002

Using AT Server OLE Functions in Visual C++

Author:	A Jesson						
Issue:	1						
Date:	18 July 00						

For further applications, sales and service advice contact your local supplier or Voltech at:

Voltech Instruments Inc.

Tel. +1 919 431 0015
Fax. +1 919 431 0090

sales@voltech.com
http://www.voltech.com

Voltech Instruments Ltd.

Tel. +44 1235 834555
Fax. +44 1235 835016

sales@voltech.co.uk
http://www.voltech.co.uk

USA, Canada,
Central & South America

Voltech™

Europe, Asia,
Middle East & Africa

Introduction

Using the AT Series Server OLE2 functions in Visual C++ is much more involved than when using Visual Basic (or Visual Basic for Applications in Excel).

This Application Note will give some basic guidance to those with some knowledge of Visual C++, but have not yet attempted an Automation Client Application before.

You will require the "Server.tlb" type library file, available from Voltech Instruments. Please contact your Voltech supplier.

This technical note has been compiled using Microsoft Visual C++ version 6. There may be differences when using other development environments.

Setting up Visual C++ Application for OLE

Set-up your new application using the Program Wizard as normal, creating a Dialog, SDI or MDI MFC executable program.

To allow for Automation Clients to use OLE functions, all MFC OLE applications require the following call to `AfxOleInit`, which initialises the OLE DLLs so that they can call OLE interfaces. Add the following code to the `InitInstance` member function of your applications' CWinApp-derived class:

```
if (!AfxOleInit())
{
     AfxMessageBox(IDP_OLE_INIT_FAILED);
     return FALSE;
}
```

You will also have to create a string on your string table for `IDP_OLE_INIT_FAILED`, something like:

```
"OLE initialisation failed. Make sure that the OLE libraries are
the correct version."
```

Setting up the OLE Interface Class for the Server OLE Functions

To set up the Interface Class for the Server OLE Function you need to use the Class Wizard. Go to the Class Wizard as normal using the top menus (View > Class Wizard) or CTRL+W.

Select the Automation tab and then go to Add Class button. From here select 'From a Type Library'. You then get a browser dialog box appear and you should select the server.tlb file, where ever you have placed this file, and select open.

You can check and confirm the classname of the interface class, making sure that they do not clash with your current project names.

The default Class name is 'IAT3600LinkDocument'. The base class is COleDispatchDriver, which allows all OLE functions to work from.

The header and implementation files can be changed from the default server.h and server.cpp. It would be more useful to rename these as ATLink.h and ATLink.cpp respectively.

When all the details have been checked the class is created and the header and implementation files added to the project. When the interface class is created, it sets up each individual InvokeHelper command for each individual OLE function.

Setting up and performing OLE functions

Set-up the use of the new Interface Class for performing the OLE functions of the Server by adding the "ATLink.h" header file to your dialog or main program. Add the following to your header file's implementation section, something like:

```
// Implementation
protected:
      IAT3600LinkDocument m_iServerResults;
```

You may call your member variable anything you like, but the above follows a default, which will be used later. This sets up your interface variable and you can perform all the functions necessary for OLE communications through it. When you need to perform OLE communication with the Server, you will need to add the following code to your program:

```
...
COleException oe;
BOOL b;
CString sh;
...
b = m_iServerResults.CreateDispatch(("Server.Results"), &oe);
if (!b)
{
      sh.Format("Error trying to open Server.Results");
      Messagebox(sh, "Automation/OLE", MB_ICONERROR | MB_OK);
}
```

This sets the variables for using the CreateDispatch function (which is about the same as using CreateObject in Visual Basic). It also traps any errors if there is a problem creating the dispatch object.

When you have finished using OLE communication, eg when the program has finished, you must always release the dispatch with the following:

```
m_iServerResults.ReleaseDispatch();
```

To use the functions available to you to get data, you will have to use a VARIANT data type. Use the Help function to get a full understanding of the variant data type in Visual C++. You may want to set up a public member variant in the header file, but do not forget to initialise the variant with the following code:

```
VARIANT vntResult;
VariantInit(&vntResult);
```

For details of how to check a variant for internal data type and for getting values and strings in the variant type, see the Visual C++ help system. To use the available OLE functions, you can perform commands like:

```
vntResult = m_iServerResults.GetVariantResult("COM2",1);
```

or

```
vntResult = m_iServerResults.GetUnitType("COM2");
```

Available Functions

The current list of commands and attributes are as follows:

ID Code	Function	Return Variable
0x01 (1)	GetResult (LPCSTR ComPort, short TestNumber)	float
0x02 (2)	GetVariantResult (LPCSTR ComPort, short TestNumber)	Variant
0x03 (3)	GetPartID (LPCSTR ComPort)	Variant
0x04 (4)	GetFixtureID (LPCSTR ComPort)	Variant
0x05 (5)	GetOperatorID (LPCSTR ComPort)	Variant
0x06 (6)	GetBatchID (LPCSTR ComPort)	Variant
0x07 (7)	GetCompensationStatus (LPCSTR ComPort)	Variant
0x08 (8)	GetFirmwareID (LPCSTR ComPort)	Variant
0x09 (9)	GetUnitID (LPCSTR ComPort)	Variant
0x0A (10)	GetTransformerSerialNo (LPCSTR ComPort)	Variant
0x0B (11)	GetResultsFilename (LPCSTR ComPort)	Variant
0x0C (12)	GetNumberOfTests (LPCSTR ComPort)	Variant
0x0D (13)	GetMinLimit (LPCSTR ComPort, short TestNumber)	Variant
0x0E (14)	GetMaxLimit (LPCSTR ComPort, short TestNumber)	Variant
0x0F (15)	GetResultPolarity (LPCSTR ComPort, short TestNumber)	Variant
0x10 (16)	GetOverallPass (LPCSTR ComPort)	Variant
0x11 (17)	GetTestPass (LPCSTR ComPort, short TestNumber)	Variant
0x12 (18)	GetTestStatus (LPCSTR ComPort, short TestNumber)	Variant
0x13 (19)	GetTestMnemonic (LPCSTR ComPort, short TestNumber)	Variant
0x14 (20)	GetTestUnits (LPCSTR ComPort, short TestNumber)	Variant
0x15 (21)	GetPolarityPass (LPCSTR ComPort, short TestNumber)	Variant
0x16 (22)	NewDataAvaiable (LPCSTR ComPort)	Variant
0x17 (23)	GetUnitType (LPCSTR ComPort)	Variant
0x18 (24)	LoadProgramToAT (LPCSTR ComPort, LPCSTR ProgName)	Variant
0x19 (25)	RunProgramInAT (LPCSTR ComPort)	Variant
0x1A (26)	StopAT (LPCSTR ComPort)	Variant

Notes:

1. NewDataAvailable function does not return a variant boolean type. This function returns (if applicable) a short INT variant (VT_I2) with 0 for FALSE and 1 for TRUE.

2. To reset the NewDataAvailable flag, you must collect at least one GetVariantResult or GetResult.

3. The GetResult function returns a true float data type. It is used only for backward compatibility with older software and should not be used for new programs. GetVariantResult, only, should be used to collect results in new code.

Timing Sequence of results returned from Server

The results that are available from the OLE functions depend on what happens both in the server and the transformer test unit.

There will be error messages if the Server software has not been installed on the computer. The reason for this is that during the installation of the Server software, the OLE functionality is registered in the Windows Registry.

There will also be an error if the Server software is not running when the created Automation Client application is running (your program).

To start getting any results, you must ensure that the Server application is open, and you have opened the COM Port channel that the AT Series unit is connected to. You will still get no results (except fail messages) until a program has been loaded into the AT. This can be done via the front panel of the AT or via the OLE command LoadProgramToAT.

Until a program is loaded, only information about the AT tester can be gathered (ie. UnitID, FirmwareID). All other data, including GetNumberOfTests, will only bring back valid results once the test program has been run at least once.

Note: Your test program must be set up with the option of "Send Results to Server".