DispHelper

DispHelper Version 0.81 - July 2004.

Introduction

The aim of the DispHelper library is to allow C/C++ programmers to use COM objects in a simple, script like manner.

If you did not download DispHelper from Sourceforge, please visit the DispHelper project page to get the latest version.

You can browse the DispHelper source code and samples online.

Basic Sample

```
	dhCreateObject(L"Word.Application", NULL, &wdApp);
	dhPutValue(wdApp, L".Visible = %b", TRUE);
	dhCallMethod(wdApp, L".Documents.Add");
	dhCallMethod(wdApp, L".Selection.TypeText(%s)", "DispHelper Sample");
```

Compatibility

DispHelper has been successfully compiled with Dev-C++, Visual C++ and LCC-WIN32. It is written to be compatible with as many different Windows C/C++ compilers as possible. If you have trouble using it with a Windows compiler please provide feedback so that I can attempt to rectify the problem.

A compiled version of DispHelper will run on Windows 95, 98, Me, NT4, 2000, XP, 2003 and subsequent operating systems(including 64 bit versions).

Usage

DispHelper is distributed as a multi file source and a single file version. It is recommended that you use the single file source(disphelper.c) in your projects.

To begin using DispHelper, include **disphelper.c** and **disphelper.h** in your project. These files are available in the **single_file_source** directory.

For Visual C++, Borland C++ and LCC-Win32 import libraries are included via pragma directives. For other compilers you may need to add **ole32**, **oleaut32** and **uuid**.

To do this in Dev-C++ add **"-lole32 -loleaut32 -luuid"** to the **linker** box under **Project->Project Options->Parameters**.

If you are using Dev-C++ and get errors when compiling disphelper.c please make sure disphelper.c is **set to compile as C** and not C++ under **Project->Project Options->Files**.

Distribution

DispHelper is provided under the BSD licence. This license allows you to distribute DispHelper in binary and/or source form with your commercial or non-commercial projects.

Feedback, Bugs and Support

All feedback, bug reports, support requests, criticism, etc are very welcome. You can deliver feedback via the DispHelper forum or post a support request.

Stability

DispHelper should be considered fairly stable. I have used it for several months, including in commercial projects, and analysed the source code for bugs without uncovering any issues.

Directory Layout

source: Contains the source code for DispHelper.
single_file_source: Contains a compacted single file version of the source for easy inclusion in your

projects.
samples_c: Contains several DispHelper samples in C.
samples_cpp: Contains several DispHelper samples in C++.
samples_applications: Currently empty. Will contain small sample applications that make use of DispHelper.

Samples

Several samples are available in the **samples_c** and **samples_cpp** directories. Batch files to compile these samples are provided for Dev-C++, Visual C++ and LCC-WIN32. Drag the sample file you wish to compile onto the appropriate batch file. You may have to edit the batch file for your environment.

While each of the samples in samples_c will compile as either C or C++, more C++ friendly versions, which make use of smart pointers, C++ exceptions and the C++ standard libraries are available in the samples_cpp directory.

Sample List

File	Description
word	Demonstrates outputting formatted text to a Word document and getting user feed back with the help of the office assistant. Demonstrates using Word as a spell checker.
excel	Demonstrates outputting formatted data to Excel and using it to create a chart. Demonstrates using a safe array to efficiently insert data into Excel.
email	Demonstrates sending an email with CDO, Outlook and Eudora.
ado	Demonstrates reading and manipulating data from a data source using ActiveX Data Objects.
corel	Demonstrates outputting formatted text to a WordPerfect document.
speech	Demonstrates using Microsoft Agent and SAPI to provide text-to-speech.
MSHTML	Demonstrates ui-less html parsing and manipulation of the html document object model (DOM) using MSHTML. Provides functions to parse html from a string, a website or a file.
regexp	Demonstrates using the VBScript.RegExp object to provide support for regular expressions. Provides a function to extract hrefs from a web page using a regular expression.
xml	Demonstrates using MSMXL to download a web page, read an RSS feed and read and manipulate XML with the XML document object model(DOM).
wmi	Demonstrates using Windows Management Instrumentation(WMI). Samples include enumerating installed hot fixes, purging print queues, starting a program, monitoring starting and terminating processes, and monitoring file creation. All of these samples can target the local or a remote computer.
pocketsoap	Demonstrates using the PocketSoap toolkit to utilise several web services. Play the 'Who wants to be a millionaire' quiz with the help of a web service.
soap	Demonstrates using the MSSoap toolkit to utilise several web services. Web services demonstrated include Google search, spell checker and cache viewer.
iexplore	Demonstrates controlling Internet Explorer via COM. Demonstrates using an Internet Explorer window to display or retrieve information from the user.
scriptctl	Demonstrates using the MSScriptControl to run a VBScript or JScript.
dexplore	Demonstrates controlling Microsoft's new help system for developers, dexplore.
dcom_alt_creds	Demonstrates one way of accessing a remote COM object using alternate credentials.
wia	Demonstrates using Windows Image Acquisition(WIA) to manipulate images. Demonstrates taking a snapshot from a video device.

Current Limitations

The time_t conversion routines used by the "%t" identifier make use of the mktime(), localtime() and

gmtime() functions. These functions may not be thread safe on some compilers.

Micro Tutorial

Let's look at a very simple program. This program will grab the web page at sourceforge.net and dump it to the console. Error handling is omitted for simplicity.

```
01  #include "disphelper.h"
02  #include <stdio.h>
03
04  int main(void)
05  {
06      DISPATCH_OBJ(objHTTP);
07      LPSTR szResponse;
08
09      dhInitialize(TRUE);
10      dhToggleExceptions(TRUE);
11
12      dhCreateObject(L"MSXML2.XMLHTTP", NULL, &objHTTP);
13      dhCallMethod(objHTTP, L".Open(%s, %s, %b)", "GET", "http://sourceforge.net", FALSE);
14      dhCallMethod(objHTTP, L".Send");
15
16      dhGetValue(L"%s", &szResponse, objHTTP, L".ResponseText");
17
18      printf("Response:\n%s\n", szResponse);
19      dhFreeString(szResponse);
20
21      SAFE_RELEASE(objHTTP);
22      dhUninitialize(TRUE);
23  }
```

On line **01** we include disphelper.h. Because DispHelper uses late bound COM we do not have to include a header file for a specific COM object.

On line **04** we start a normal console project. DispHelper can be used in both console projects and GUI projects.

On line **06** we declare a COM object for use with DispHelper functions. The DISPATCH_OBJ macro is declared as such:

```
#define DISPATCH_OBJ(objName) IDispatch * objName = NULL
```

Line **09** initializes DispHelper for this thread. By passing TRUE to dhInitialize() it will also initialize COM for this thread.

Line **10** instructs DispHelper to display errors. Any error that occurs during a call to a DispHelper function will be displayed in a message box. This feature is very useful for debugging.

On line **12** we create our COM object. The first argument to dhCreateObject() specifies which component we wish to create, in this case an XMLHTTP object. The second argument specifies which computer the COM object should be created on. COM objects that run in their own process, such as Word and Excel can be created on a remote computer, assuming you have access rights. We pass NULL to create the object on the local computer. Finally, we pass the address of objHTTP to receive the created COM object.

Lines **13 and 14** show our first method calls. As you can see, DispHelper uses a printf style syntax. ".Open(%s, %s, %b)" means that we want to call the "Open" method with 2 strings and a BOOL as parameters. Line **14** calls the "Send" method with no arguments.

Line **16** shows how to retrieve a value. The value can be from either a property or the return value of a function. The first argument passed to dhGetValue() is an identifier which specifies the type to return. In this case we pass "%s" to specify we want the value as a string. The second argument passes the address of where we want the return value. This must match the type specified in the first argument. We pass the address of a string. Subsequent arguments are the same as for dhCallMethod().

Line **18** prints the returned string to the console. Line **19** frees the returned string. Strings returned by dhGetValue() must be freed using dhFreeString().

Line **21** releases our COM object. COM objects returned by DispHelper functions should be freed using the SAFE_RELEASE() macro.

You may be wondering what all those Ls dotting the code are for. An L before a string literal in C and C++ specify that the string literal should be a unicode string literal. Due to the fact that COM typically uses unicode strings most DispHelper string arguments need to be passed as unicode strings. On the other hand, strings passed to methods you are calling on an object can be either normal strings(%s) or wide unicode strings(%S).

Format Identifiers

As you have seen, DispHelper functions uses format identifiers to specify argument types. Below is a complete list of the format identifiers that DispHelper handles.

Identifier	Type	Compatible Types	Note
%d	LONG	long, int, INT	
%u	ULONG	unsigned long, unsigned int, UINT, DWORD	
%e	DOUBLE	double	
%b	BOOL		
%v	VARIANT		
%B	BSTR		1
%s	LPSTR	char *	1
%S	LPWSTR	WCHAR *	1
%T	LPTSTR	TCHAR *	1
%o	IDispatch *	DISPATCH_OBJ(var)	2, 3
%O	IUnknown *		3
%t	time_t		4
%W	SYSTEMTIME *		4
%f	FILETIME *		4
%D	DATE		4
%p	LPVOID	Use for HANDLEs, HWNDs, etc	
%m	Missing Argument		5
Note 1: When a string is returned using dhGetValue() it should be freed using dhFreeString().			
Note 2: When an object is declared using DISPATCH_OBJ(var) it is an IDispatch *.			
Note 3: When an IDispatch * or IUnknown * is returned from dhGetValue() it should be released using the SAFE_RELEASE() macro.			
Note 4: A variant DATE(%D) is always in local time. When passing in or receiving a FILETIME(%f) or SYSTEMTIME(%W) no time zone translation is performed. Therefore, you should pass in and expect to receive SYSTEMTIMEs and FILETIMEs in local time. However, as a time_t(%t) is always in GMT time, time zone translation is performed by DispHelper.			
Note 5: %m can only be used as an input argument. It specifies a missing optional argument. It does not have a corresponding entry in the argument list. `dhCallMethod(myObj, L".DoSomething(%m, %s)", "test");`			

Quick Reference

Initializing DispHelper

```
// Initializes DispHelper and COM for this thread
dhInitialize(TRUE);

// Initializes DispHelper only. COM must be initialized seperately
// using CoInitialize(), CoInitializeEx() or OleInitialize().
dhInitialize(FALSE);
```

```
// Using the C++ initialization class
CDhInitialize init;
```

Declaring a DispHelper COM object

```
// With the macro
DISPATCH_OBJ(wdDoc);

// Without the macro
IDispatch * wdDoc = NULL;

// Using the C++ smart pointer
CDispPtr wdDoc;
```

Toggling Exceptions

```
// DispHelper will show a message box when an error occurs. Useful for debugging.
dhToggleExceptions(TRUE);

// DispHelper will not show a message box when an error occurs(default).
dhToggleExceptions(FALSE);
```

Creating a COM object

This function is similar to the VBScript function CreateObject().

```
// Create an ADO connection object.
dhCreateObject(L"ADODB.Connection", NULL, &adoConn);

// Create a word document on the computer "server"
dhCreateObject(L"Word.Document", L"server", &wdDoc);
```

Getting a COM object

This function is similar to the VBScript function GetObject().

```
// Get a running instance of word
dhGetObject(NULL, L"Word.Application", &wdApp);

// Load "my word doc.doc" as a COM object
dhGetObject(L"my word doc.doc", L"Word.Document", &wdDoc);
```

Calling a method

```
// Type some text in Word
dhCallMethod(wdApp, L".Selection.TypeText(%s)", "Some text");

// Save the 2nd Word document
dhCallMethod(wdApp, L".Documents(%d).Save", 2);
```

Setting a property value

```
// Make Word visible.
dhPutValue(wdApp, L".Visible = %b", TRUE);

// Set the age field of an ADO recordset
dhPutValue(adoRs, L".Fields(%S).Value = %d", L"Age", 34);
```

Retrieving a value

```
// Get the text of a word document
dhGetValue(L"%s", &szText, wdDoc, L".Range.Text");

// Get the value of an Excel cell
dhGetValue(L"%d", &nValue, xlApp, L".ActiveSheet.Range(%s).Value", "A1");

// Get the result of executing a wmi query - a collection object.
dhGetValue(L"%o", &colQuickFixes, wmiSvc, L".ExecQuery(%S)",
```

```
                                    L"SELECT * FROM Win32_QuickFixEngineering");
```

Enumerating a collection

This is made simple with a couple of macros.

```
    // The enumeration begins with FOR_EACH
    // objQuickFix does not need to be pre-declared.
    FOR_EACH(objQuickFix, /* in */ colQuickFixes, NULL)
    {
        // In the for each we can use objQuickFix
        dhGetValue(L"%s", &QuickFix.szDescription, objQuickFix, L".Description");
        dhGetValue(L"%s", &QuickFix.szInstalledBy, objQuickFix, L".InstalledBy");

        printf("Description: %s\nInstalled By: %s\n",
                QuickFix.szDescription, QuickFix.szInstalledBy);

        dhFreeString(QuickFix.szDescription);
        dhFreeString(QuickFix.szInstalledBy);

        // The NEXT takes care of cleaning up the enumeration
    } NEXT(objQuickFix);

    // Enumerate word documents
    FOR_EACH(wdDoc, wdApp, L".Documents")
    {
        dhCallMethod(wdDoc, L".Range.InsertBefore(%s)", "DispHelper");

    } NEXT(wdDoc);

    // The FOR_EACH macro can also take arguments
    // Note that we append FOR_EACH with the number of arguments, up to 4
    // Enumerate each word in the second document
    FOR_EACH1(wdWord, wdApp, L".Documents(%d).Words", 2)
    {
        dhPutValue(wdWord, L".Font.Bold = %b", TRUE);

    } NEXT(wdWord);
```

Optimising with WITH

With a similar syntaxt to the FOR_EACH macro the WITH macro allows you to avoid retrieving an object multiple times.

```
    // Without the WITH macro
    dhPutValue(wdApp, L".Selection.Font.Size = %d", 20);
    dhPutValue(wdApp, L".Selection.Font.SmallCaps = %b", TRUE);
    dhPutValue(wdApp, L".Selection.Font.Name = %S", L"Comic Sans MS");
    dhPutValue(wdApp, L".Selection.Font.Color = %d", RGB(33,99,0) );

    // Optimized with the WITH macro
    WITH(wdFont, wdApp, L".Selection.Font")
    {
        dhPutValue(wdFont, L".Size = %d", 20);
        dhPutValue(wdFont, L".SmallCaps = %b", TRUE);
        dhPutValue(wdFont, L".Name = %S", L"Comic Sans MS");
        dhPutValue(wdFont, L".Color = %d", RGB(33,99,0) );

    } END_WITH(wdFont);

    // The WITH macro can also take arguments.
    WITH1(wdFont, wdApp, L".Documents(%d).Range.Font", 1)
    {
        dhPutValue(wdFont, L".Size = %d", 20);
        dhPutValue(wdFont, L".SmallCaps = %b", TRUE);

    } END_WITH(wdFont);
```

Retrieving the last error text

```
    // Dump the details of the last error to a log file
    void LogDhError(void)
    {
        char szMessage[512];
        dhFormatExceptionA(NULL, szMessage, sizeof(szMessage)/sizeof(szMessage[0]), TRUE);
        fprintf(g_hLogFile, "%s", szMessage);
    }

    // Sample usage
    if (FAILED(dhCallMethod(wdApp, L".Selection.TypeText(%s)", "Hello World")))
    {
        LogDhError();
        // Cleanup and exit
    }
```

Throw a C++ exception on error

```
    // Use the dhCheck macro which will throw a std::string on error

    try
    {
        dhCheck( dhCallMethod(wdApp, L".Selection.TypeText(%s)", "Hello World") );
        dhCheck( dhPutValue(wdApp, L".Selection.Font.Bold = %b", TRUE) );
    }
    catch (string errstr)
    {
        cerr << "Fatal error details:" << endl << errstr << endl;
    }
```

Releasing a COM object

```
// Make use of the SAFE_RELEASE macro - do not use with smart pointer
SAFE_RELEASE(wdApp);
```

Future Directions

DispHelper may expand to allow easy hosting of ActiveX controls, if demand exists.