



API Integration Guide for
C/C++

WWW.SOFTWARE-SECURITY-PRODUCTS.CO.UK/PROTECTIT

PROTECTit API Integration Guide for C/C++

Please note that the C/C++ integration relies on the *IDispatch* interface to access **Protect-IT**. All code here has been tested using *Microsoft Visual C++ 2008 Express Edition*. Sample code can be found in the **Protect-IT** CD » *protectit_files* » *com_server* » *samples* directory.

Definitions

Activation Code: A code sent to a user by email (default) or forwarded by other electronic means that enables the dongle values to be changed.

Online Update: An operation whereby a connection is made to the **Protect-IT** central database and the dongle is automatically updated there and then.

Dongle 'slot': A space on the dongle to store application critical information. There are 5 'slots' available for each type of data: integer, text string, double and boolean, in both the 'admin' and 'Application' access areas, totalling 40 'slots'.

Expired Dongle: A dongle that is not valid. A dongle whose expiry date is less than the current date (non-inclusive).

Valid Dongle: A dongle that is not expired. A dongle whose expiry date is greater than or equal to the current date (as stored on the dongle).

Before You Begin the Integration

Dongle Driver Installation

Ensure that the dongle driver is correctly installed by running the '*Sentinel System Driver Installer 7.5.0.exe*'. A silent installation of the dongle driver can be performed by using the following command line (all on one line without line breaks):

```
"Sentinel System Driver Installer 7.5.0.exe" /s /v"/q  
ADDLOCAL=USB_Driver CONFIRMUPGRADE=TRUE"
```

Or by executing the batch file:

```
dongle_driver_silent_install.bat
```

All dongle related files can be found in the **Protect-IT** CD » *dongle_driver* directory.

Java Runtime Environment (JRE)

Java Runtime Environment (JRE) 1.6 and above is required to run the **Protect-IT API**.

Please download and install the latest version, which can be found at: www.java.com

Deployment File Structure

All files required can be found in the **Protect-IT** CD » *protectit_files* » *com_server* directory.

Ensure your deployment structure contains the following: 

Ensure that within your installation directory, there is a directory called *lib* that contains all the jar files: *protectit.jar*, *comfjy-2.5.jar*, *comfjy-svrmanager-2.5.jar*, *jniwrap-3.7.jar* and *winpack-3.6.jar*.

Ensure that within your installation directory the *register_com_server.bat* and *unregister_com_server.bat* files are present.

Registering the Protect-IT COM Server

In order to access **Protect-IT** you will need to register the **Protect-IT** COM Server.

This procedure needs to be done only once by running the *register_com_server.bat* file from your installation directory (from a command prompt with administration rights for *Windows Vista*).

Your Client_id

Before you can begin integrating, you will also need your *client_id* given to you with your first dongle. The *client_id* needs to be included in your code for **Protect-IT** to correctly recognise your dongles.

Quick Integration Guide

This is how to use C/C++ with **Protect-IT**. The code listed here is only a basic guide. The **Protect-IT COM Server** extends the standard *IDispatch* interface so you may want to use your own code to access it.

This is the order of events that **must** be followed to get **Protect-IT** correctly integrated with your software application.

- Initialise the **Protect-IT COM Server** and get a pointer to the *IDispatch* interface. (page 5) All **Protect-IT** method calls **must** use the *IDispatch* pointer and so it is recommended that a globally available reference to this pointer is kept.
- Initialise **Protect-IT**. This allows **Protect-IT** it to confirm that an appropriate dongle is plugged in. (page 6)
- Select a valid dongle. (page 7)
- Use the **Protect-IT** methods to access the dongle. (page 8)

The following libraries have been used in code:

```
#include <stdio.h>
#include <tchar.h>
#include <comutil.h>
```

To make the code in this booklet shorter, the following method for invoking **Protect-IT** methods is used throughout:

```
// Invokes the method named, with the passed parameters (params) and
// stores the result on the pointer &result.
HRESULT invokeMethod( IDispatch *iDispatchInst, OLECHAR
*methodName, DISPPARAMS params, VARIANT &result ) {
    DISPID dispid;
    HRESULT hr = S_OK;
    hr = iDispatchInst->GetIDsOfNames( IID_NULL, &methodName, 1,
    GetUserDefaultLCID(), &dispid);
    if ( FAILED(hr) ) {
        wprintf(L"Failed to find Method:%s. Execution failed.\n",
```

```
methodName);
        return hr;
    }
    hr = iDispatchInst->Invoke(dispid,
        IID_NULL,
        GetUserDefaultLCID(),
        DISPATCH_METHOD,
        &params,
        &result,
        NULL,
        NULL);

    if ( FAILED(hr) ) {
        wprintf(L"Method:%s execution failed!%s\n", methodName);
    }
    return hr;
}
```

The following variables are also required:

```
DISPPARAMS params; // The parameters to pass to a method
DISPPARAMS noParams = {NULL, NULL, 0, 0,}; // No parameters
VARIANT result; // The result from executing the method
```

Initialise the Protect-IT COM Server

The *COM Server* implements the default *IDispatch* interface.

```
HRESULT hr = S_OK;
// Initialize COM.
CoInitialize(NULL);
// Use Protect-IT Program ID "com.ssp.protectit.com"
// or its CLSID "{D7E92A04-BD06-40fb-B48B-B3F10E5B36DF}"
LPCOLESTR progID = OLESTR("com.ssp.protectit.com");
LPCLSID clsID = new CLSID;
CLSIDFromProgID(progID,clsID);
// Working with the COM Server via the IDispatch interface
IDispatch* iDispatchInst;
```

```

hr = CoCreateInstance(*clsID,NULL,CLSCTX_LOCAL_SERVER,
    IID_IDispatch, (void **)&iDispatchInst);
if ( SUCCEEDED(hr) ) {
    printf("Protect-IT successfully found.\n");
}
else {
    printf("Error finding Protect-IT.\n");
    printf("Have you run register_com_server.bat?\n");
    return 1;
}

```

The *HRESULT* object needs to be checked against the *SUCCEEDED(hr)* or *FAILED(hr)* methods at every invocation. A COM Exception is thrown corresponding to a *HRESULT* of *E_UNEXPECTED* (captured as *FAILED(hr) == true*) to indicate that the application should be exited.

E_UNEXPECTED is thrown by the **Protect-IT API** when the user has failed to attach a valid dongle, has removed one whilst the application is in use, has an expired dongle or has tampered with the system clock.

Initialise Protect-IT

For **Protect-IT** to confirm that an appropriate dongle is plugged in, the method *'initialise'* must be invoked with the *client_id*.

Alternatively, if the application has an *application_id* other than the default value of 0 (zero), the method *'initialiseForApplication'* must be invoked with the *client_id* and *application_id*. (Please refer to *Appendix A* for an example of its usage)

Please note that you must use one of the two methods above before proceeding to the next step.

```

// Calling the "initialise" method with client_id 100
DISPID dispid;
OLECHAR* methodName;
VARIANTARG* arguments = new VARIANTARG[1];

```

```

VARIANTARG vargNew;
VariantInit(&vargNew);
vargNew.vt = VT_I8;
vargNew.intVal = 100; // client_id
arguments[0] = vargNew;

```

```

DISPPARAMS params;
params.cArgs = 1;
params.rgvarg = arguments;
params.cNamedArgs = 0;
params.rgdispidNamedArgs = NULL;

```

```

OLECHAR* methodName = L"initialise";
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,methodName,params,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}

```

Selecting a Valid Dongle

In order to start using the program a valid dongle (that is, a dongle that has not expired) must be plugged-in. If the dongle that is plugged-in has expired (or is about to expire) the user will be given the opportunity to apply an *Activation Code* or perform an *Online Update*.

```

// Calling the "selectedValidDongle" method
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,L"selectValidDongle",noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}

```

If a valid dongle is not found (even after applying an *Activation Code* or *Online Update*) the application will not be started and the user has the option to exit at this stage, otherwise, normal application operation continues.

Protect-IT Methods to Access the Dongle

The dongle is capable of storing various types of data. **Protect-IT** allows you to save the following data types:

- **String** – Text of no more than 32 characters.
- **Integer** – A whole (positive or negative) number between 0 and 65,535.
- **Double** – A (positive or negative) number to the accuracy of 3 decimal places between 0 and 4,294,967.295.
- **Boolean** – Stores a value of either true (on) or false (off).

There are 5 ‘slots’ on the dongle for each of the above data types in both the ‘Admin’ and ‘Application’ access areas.

Admin Access Area

This is an area of the dongle that you (as the *Administrator*) have full control of and should be used for application configuration, enabling/disabling modules, etc. The *Admin Access Area* values **can not** be modified programmatically. *Admin Access Area* values can only be updated through *Activation Codes* and *Online Updates*.

Application Access Area

This is an area of the dongle that your application has full control of and should be used for data that is relevant to the working of the application, but which you (as an *Administrator*) do not have direct access to. It could be used for storing user settings for your application, application counters, etc.

Online Updates and *Activation Codes* **do not** update the *Application Access Area*.

Accessing Your Dongle Data

You have the ability to programmatically read and write different types of data. The table opposite describes how each bit of data can be used programmatically or by the application of an *Activation Code* (AC) or conducting an *Online Update* (OU).

Property Name	Programmatically		AC/OU	Description
	Read	Write	Write	
Client Id	✓	✗	✗	Your unique <i>Client_id</i> sent to you with your dongles.
(Optional) Application Id	✓	✗	✓	(Optional) use it if you have more than one application to protect.
Dongle Serial	✓	✗	✗	The unique dongle id.
Expiry Date	✓	✗	✓	Controls the expiry mechanism for your application.
(Optional) Username	✓	✗	✓	(Optional) use it to give a meaningful name to each dongle. E.g. <i>Client 1</i> , <i>Client 2</i> .
Language Code	✓	✗	✓	A 2 characters-long code (ISO-639-1) used for the automatic display of internationalised messages to the user (can also be used internally by your application).
(Optional) Country Code	✓	✗	✓	(Optional) A 2 characters-long code (ISO-3166) used for the automatic display of internationalised messages to the user (can also be used internally by your application).
Integer 1–5	✓	✗	✓	5 ‘slots’ on the dongle for storing whole (positive or negative) numbers between 0 and 65,535. (<i>Admin Access Area</i>)
String 1–5	✓	✗	✓	5 ‘slots’ on the dongle for storing text of no more than 32 characters. (<i>Admin Access Area</i>)
Double 1–5	✓	✗	✓	5 ‘slots’ on the dongle for storing (positive or negative) numbers to the accuracy of 3 decimal places between 0 and 4,294,967.295. (<i>Admin Access Area</i>)
Boolean 1–5	✓	✗	✓	5 ‘slots’ on the dongle for storing values of either true (on) or false (off). (<i>Admin Access Area</i>)
Application Integer 1–5	✓	✓	✗	5 ‘slots’ on the dongle for storing whole (positive or negative) numbers between 0 and 65,535. (<i>Application Access Area</i>)

TABLE CONTINUES ON NEXT PAGE

Property Name	Programmatically		AC/OU	Description
	Read	Write	Write	
Application String 1-5	✓	✓	✗	5 'slots' on the dongle for storing text of no more than 32 characters. (Application Access Area)
Application Double 1-5	✓	✓	✗	5 'slots' on the dongle for storing (positive or negative) numbers to the accuracy of 3 decimal places between 0 and 4,294,967.295. (Application Access Area)
Application Boolean 1-5	✓	✓	✗	5 'slots' on the dongle for storing values of either true (on) or false (off). (Application Access Area)

Reading Integers

The following methods can be used for programmatically reading integers from the dongle:

readClientId, *readApplicationId*, *readSerial*, *readInteger1*, *readInteger2*, *readInteger3*, *readInteger4*, *readInteger5*, *readApplicationInteger1*, *readApplicationInteger2*, *readApplicationInteger3*, *readApplicationInteger4*, *readApplicationInteger5*.

An example using method *readSerial* is given below:

```
// Calling the "readSerial" method
OLECHAR* methodName = L"readSerial";
HRESULT hr = S_OK;
hr =
invokeMethod(iDispatchInst,L"readExpiryDate",noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
else {
    int serial = result.intVal;
    wprintf(L"Serial = %i\n", serial );
}
```

Writing Integers

The following methods can be used for programmatically writing integers to the dongle:

writeApplicationInteger1, *writeApplicationInteger2*, *writeApplicationInteger3*, *writeApplicationInteger4*, *writeApplicationInteger5*.

An example using method *writeApplicationInteger1* is given below:

```
// Calling the "writeApplicationInteger1" method to write 5
VARIANTARG* arguments = new VARIANTARG[1];
VARIANTARG vargNew;
VariantInit(&vargNew);
vargNew.vt = VT_I8;
vargNew.intVal = 5; // value to write
arguments[0] = vargNew;
DISPPARAMS params;
params.cArgs = 1;
params.rgvarg = arguments;
params.cNamedArgs = 0;
params.rgdispidNamedArgs = NULL;
```

```
OLECHAR* methodName = L"writeApplicationInteger1";
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,methodName,params,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
```

Reading Strings

The following methods can be used for programmatically reading strings from the

dongle: *readExpiryDate*, *readUsername*, *readLanguageCode*, *readCountryCode*, *readString1* to *readString5*, *readApplicationString1* to *readApplicationString5*.

An example using method *readExpiryDate* is given below:

```
// Calling the "readExpiryDate" method
```

```

HRESULT hr = S_OK;
hr =
invokeMethod(iDispatchInst,L"readExpiryDate",noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
else {
    wprintf(L"Expiry Date (yyyymmdd) = %s\n", result.bstrVal );
}

```

Writing Strings

The following methods can be used for programmatically writing strings to the dongle: *writeApplicationString1* to *writeApplicationString5*.

An example using method *writeApplicationString1* is given below:

```

// Calling the "writeApplicationString1" method to write "abc"
BSTR aString = SysAllocString(L"abc");
VARIANTARG* arguments = new VARIANTARG[1];
VARIANTARG vargNew;
VariantInit(&vargNew);
vargNew.vt = VT_BSTR;
vargNew.bstrVal = aString; // value to write
arguments[0] = vargNew;

```

```

DISPPARAMS params;
params.cArgs = 1; // The count of arguments
params.cNamedArgs = 0; // The count of named arguments
params.rgdispidNamedArgs = NULL; // The dispatch IDs of named
arguments
params.rgvarg = arguments; // A referenece to the array of
arguments
OLECHAR* methodName = L"writeApplicationString1";
HRESULT hr = S_OK;

```

```

hr = invokeMethod(iDispatchInst,methodName,params,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}

```

Reading Doubles

The following methods can be used for programmatically reading doubles from the dongle: *readDouble1* to *readDouble5*, *readApplicationDouble1* to *readApplicationDouble5*.

An example using method *readDouble1* is given below:

```

// Calling the "readDouble1" method
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,L"readDouble1",noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
else {
    wprintf(L"Double1 = %f\n", result.dblVal );
}

```

Writing Doubles

The following methods can be used for programmatically writing doubles to the dongle: *writeApplicationDouble1* to *writeApplicationDouble5*.

An example using method *writeApplicationDouble1* is given below:

```

// Calling the "writeApplicationDouble1" method with write
"12.345"
double aDouble = 12.345;
VARIANTARG* arguments = new VARIANTARG[1];
VARIANTARG vargNew;
VariantInit(&vargNew);
vargNew.vt = VT_R8;

```

```
vargNew.dblVal = aDouble; // value to write
arguments[0] = vargNew;
```

```
DISPPARAMS params;
params.cArgs = 1;
params.rgvarg = arguments;
params.cNamedArgs = 0;
params.rgdispidNamedArgs = NULL;
```

```
OLECHAR* methodName = L"writeApplicationDouble1";
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,methodName,params,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
```

Reading Booleans

The following methods can be used for programmatically reading booleans from the dongle: *readBoolean1* to *readBoolean5*, *readApplicationBoolean1* to *readApplicationBoolean5*.

An example using method *readBoolean1* is given below:

```
// Calling the "readBoolean1" method
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,L"readBoolean1",noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
else {
    if ( result.boolVal ) printf( "Boolean1 = true" );
    else printf( "Boolean1 = false" );
}
```

Writing Booleans

The following methods can be used for programmatically writing booleans to the dongle: *writeApplicationBoolean1* to *writeApplicationBoolean5*.

An example using method *writeApplicationBoolean1* is given below:

```
// Calling the "writeApplicationBoolean1" method with write "true"
// Use VARIANT_TRUE or VARIANT_FALSE
VARIANT_BOOL b = VARIANT_TRUE;
VARIANTARG* arguments = new VARIANTARG[1];
VARIANTARG vargNew;
VariantInit(&vargNew);
vargNew.vt = VT_BOOL;
vargNew.boolVal = b; // value to write
arguments[0] = vargNew;
```

```
DISPPARAMS params;
params.cArgs = 1;
params.rgvarg = arguments;
params.cNamedArgs = 0;
params.rgdispidNamedArgs = NULL;
```

```
OLECHAR* methodName = L"writeApplicationBoolean1";
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,methodName,params,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
```

Updating the Dongles

Activation Codes and *Online Updates* can be generated and distributed using **Protect-IT's** central online database, accessible from any internet enabled browser.

Handling Errors

All error handling is automated. If there is a problem then the appropriate message will be displayed on screen where the user can choose an appropriate action.

Handling Application Exit

The default behaviour for **Protect-IT** is to throw an *E_UNEXPECTED* value to *HRESULT hr* (which can be caught by using *FAILED(hr) == true*). It is up to you as a developer to provide the required exit routine. *E_UNEXPECTED* will be returned when either:

- The user clicks on an 'Exit Application' button (within the 'dongle manager' or 'dongle expired' dialog boxes).
- The **Protect-IT COM Server** has not been registered.
- The user has not properly installed the dongle driver.
- The user failed to plug-in a dongle (or removed the dongle whilst the application was running).
- The user modified his system clock (*Time Cheating*).

Maintaining the Security of Your Application

The **Protect-IT API** protects your application and ensures revenue by:

- only allowing your application to run on a valid (non expired dongle), giving you control over the expiry date. E.g. Only update your application's monthly/quarterly *Expiry Date* once rental fees have been received;
- strong encryption that makes it extremely difficult to extract your information, clone the dongle, record and repeat dongle communication;
- employing a 'time cheating' mechanism that will instantly make your application not work if a user turns back the clock in order to get more use (once the user changes his clock back to normal, the application will resume its normal operation).

In order to ensure that all of the above works to give your application (and revenue) the best protection, the following guidelines should be followed:

1. Only send a new *Activation Code* or *Online Update* to extend the *Expiry Date* after all payments have been received. From our experience, updating on a monthly basis has been very effective;
2. Store any critical information to the running of your application on the dongle 'slots' provided (integer, string, double, boolean). This ensures that only with a valid dongle can your application be used and also gives you absolute control on all those critical values (i.e. they can be modified by *Activation Code* or *Online Update* along with the monthly expiry date update). That is another good reason to keep the expiry dates monthly, so that it forces users to update (not only the expiry date but any other dongle setting on) their dongles often;
3. Call the 'protect' method at various important stages of your application running (and sporadically, if possible) to ensure that the user has not removed his dongle during application running.

```
// Call the 'protect' method many times during the running of your
// application, at every critical operation and sporadically through
// its normal use.
// Calling the "protect" method
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,L"protect",noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
```

Extras

You can call the *Activation Code Window* or the *Online Update Window* in code by:

```
// Calling the "showActivationCodeWindow" method
OLECHAR* methodName = L"showActivationCodeWindow";
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,methodName,noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
// Calling the "showOnlineUpdateWindow" method
OLECHAR* methodName = L"showOnlineUpdateWindow";
hr = invokeMethod(iDispatchInst,methodName,noParams,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
```

By default the local system language is used to set the language for **Protect-IT**. You can set the language used by calling the *setLocale*("language code", "country code"†):

```
// Calling the "setLocale" method with language code "en"
// and no language country
// For Spanish use language: "es", country: ""
// For Portuguese use language: "pt", country: ""
BSTR sLang = SysAllocString(L"en");
BSTR sCountry = SysAllocString(L"");
// 2 Parameters
VARIANTARG* arguments = new VARIANTARG[2];
```

* The language code is a 2 characters-long text string representing a language as listed in ISO-639-1.

† The country code is a 2 characters-long text string representing a country as listed in ISO-3166.

```
VARIANTARG vargNew;
VariantInit(&vargNew);
vargNew.vt = VT_BSTR;
vargNew.bstrVal = sLang; // The language code
arguments[0] = vargNew;
```

```
VARIANTARG varg2;
VariantInit(&varg2);
varg2.vt = VT_BSTR;
varg2.bstrVal = sCountry; // The country code
arguments[1] = varg2;
```

```
DISPPARAMS params;
params.cArgs = 2; // The count of arguments
params.cNamedArgs = 0; // The count of named arguments
params.rgdispidNamedArgs = NULL; // The dispatch IDs of named arguments
params.rgvarg = arguments; // A referenece to the array of arguments
```

```
HRESULT hr = S_OK;
hr = invokeMethod(iDispatchInst,L"setLocale",params,result);
if ( FAILED(hr) ) {
    CoUninitialize(); // Free COM resources
    return 1; // Abnormal program termination
}
```

The following languages are supported: "en" English, "es" Spanish, "it" Italian, "de" German, "pt" Portuguese.

Please ensure that the 'setLocale' method is called after 'initialise' (or 'initialiseForApplication') and before the 'selectValidDongle' methods.

Appendix A

Variable Access Using the COM Interface and C++

This is how to pass variable *v* as a single parameter to a **Protect-IT** method.

Define *v* as follows for each of the data types:

```
Integer, int v = 100; // Integer value 100
String, BSTR v = SysAllocString(L"abc"); // String value "abc"
Double, double v = 12.345; // Double value 12.345
Boolean, VARIANT_BOOL v = VARIANT_TRUE; // or VARIANT_FALSE;
```

Then,

	Integer	String	Double	Boolean
<code>VARIANTARG* args = new VARIANTARG[1]; // Only one parameter passed</code>				
<code>VARIANTARG varg;</code>				
<code>VariantInit(&varg);</code>				
<code>varg.vt =</code>	<code>VT_I8;</code>	<code>VT_BSTR;</code>	<code>VT_R8;</code>	<code>VT_BOOL;</code>
<code>varg.</code>	<code>intVal = v;</code>	<code>bstrVal = v;</code>	<code>dblVal = v;</code>	<code>boolVal = v;</code>
<code>args[0] = varg;</code>				

After this, you have to create a *DISPPARAMS* object to pass to the method invocation:

```
DISPPARAMS parameters;
parameters.cArgs = 1; // Indicate that 1 parameter is passed
parameters.rgvarg = arguments;
parameters.cNamedArgs = 0;
parameters.rgdispidNamedArgs = NULL;
```

If you want to pass 2 parameters, for example for the method *'initialiseForApplication'* where 2 integers are required:

```
VARIANTARG* args = new VARIANTARG[2];
```

```
VARIANTARG varg1;
VariantInit(&varg1);
Varg1.vt = VT_I8;
Varg1.intVal = v1; // client_id
arguments[0] = varg1;
```

```
VARIANTARG varg2;
VariantInit(&varg2);
varg2.vt = VT_I8;
varg2.intVal = v2; // application_id
arguments[1] = varg2;
```

```
DISPPARAMS parameters;
parameters.cArgs = 2; // Indicate that there are 2 parameters
parameters.rgvarg = arguments;
parameters.cNamedArgs = 0;
parameters.rgdispidNamedArgs = NULL;
```

After that just invoke the method and use the &result pointer if you expect a result from say a read operation. For example:

```
HRESULT hr = S_OK;
OLECHAR *methodName = "initialiseForApplication";

hr = iDispatchInst->GetIDsOfNames( IID_NULL, &methodName, 1,
GetUserDefaultLCID(), &dispid);
if ( FAILED(hr) ) {
wprintf(L"Failed to find Method:%s. Execution failed.\n",
methodName);
return 1; // Abnormal Termination
}
// Invoke method
hr = iDispatchInst->Invoke( dispid,
IID_NULL,
GetUserDefaultLCID(),
DISPATCH_METHOD,
&parameters,
&result,
NULL,
NULL);

if ( FAILED(hr) ) {
wprintf(L"Method:%s execution failed!\n", methodName);
}
```

Please note that the above will only work after the *IDispatch Interface* has been correctly created in the C++ code and the *COM server* registered. Please refer to the sample code in the **Protect-IT** CD » *protectit_files* » *com_server* » *samples* directory.

More Information

Online at www.software-security-products.com/protectit

Please refer to the other included guides:

- **Protect-IT Distribution Guide for C++**
- **Protect-IT User Guide**
- **Protect-IT Online Guide**

Technical Support

Americas

w: www.software-security-products.com/protectit

e: support@software-security-products.com

Europe

w: www.software-security-products.co.uk/protectit

e: support@software-security-products.co.uk

Main Offices:



Software Security Products Ltd.

The Manor House
260 Ecclesall Road South
Sheffield
South Yorkshire
S11 9PS
United Kingdom





Software Security Products Limited

The Manor House 260 Ecclesall Road South Sheffield South Yorkshire S11 9PS United Kingdom

t +44(0) 1142 217 070 **f** +44(0) 1142 218 080 **e** support@software-security-products.co.uk

w www.software-security-products.co.uk