brainbuzz.com™
Cramsession

Cramsession™ for Designing and Implementing Desktop Applications with Microsoft Visual C ++ 6.0

This Cramsession will help you to prepare for Microsoft exam 70-016: Designing and Implementing Desktop Applications with Microsoft Visual C ++ 6.0. Exam topics include Deriving the Physical Design, Establishing the Development Environment, Creating the user Interface, Creating and Managing COM Components, Deploying an Application and Maintaining and Supporting an Application.

Check for the newest version of this Cramsession
http://cramsession.brainbuzz.com/checkversion.asp?V=2451956&FN=microsoft/desktopVisualC.pdf

Rate this Cramsession
http://cramsession.brainbuzz.com/cramreviews/reviewCram.asp?cert=Visual+C%2B%2B+6%2E0+Desktop

Feedback Forum for this Cramsession/Exam
http://boards.brainbuzz.com/boards/vbt.asp?b=664

More Cramsession Resources:

Search for Related Jobs
http://jobs.brainbuzz.com/JobSearch.asp?R=&CSRE

CramChallenge - practice questions
http://www.cramsession.com/signup/default.asp#day

IT Resources & Tech Library
http://itresources.brainbuzz.com

Certification & IT Newsletters
http://www.cramsession.com/signup/

SkillDrill - skills assessment
http://skilldrill.brainbuzz.com

Discounts, Freebies & Product Info
http://www.cramsession.com/signup/prodinfo.asp

Contents:

Deriving the Physical Design

MFC Framework

MFC is a set of pre-built C++ classes that are interfaces to the Windows API. MFC also adds an optional architecture that greatly simplifies development.

Main MFC elements are:

Documents

Document object is derived from CDocument class, and provides methods to access an application's data. (See also CDocument)

Views

Views determine how the user interacts with the document's data. There can be multiple views on the same data. There are many kinds of views, each deriving from a different class, like CView, CScrollView, CFormView, etc… (See also CView)

Frame Windows

Frame windows host view windows. A frame can be the main frame or a child frame in the MDI architecture. (See also CFrameWnd)

Document templates

A document template is the link between an application's document, view and frame window, allowing them to act like a single entity. (See also Document Templates and the Document/View Creation Process)

Threads

A thread is the basic unit of execution in a Window-based application.

Application Object

The app. object is the backbone of a MFC application. It encapsulate the Win32 WinMain() function. (See also CWinApp)

Differences between Windows 95, 98 and NT

Required DLLs

Not all the required DLLs exist in each kind of OS, and not all the DLLs expose the same functionalities.

Security

An application must check NT permissions and has to display correct messages to the users.

Versioning

GetVersion returns 4 on all these OSes. Use GetVersionEx instead. Don't assume 4 as a response because on Win2000 the value is 5.

Large Drives

GetFreeDiskSpace doesn't work with FAT32 drives larger than 2Gb. Use GetFreeDiskSpaceEx instead.

System Paths and registry keys

In every OS system paths and registry keys are different. This also applies in different language versions of the same OS. Use API to read correct paths and keys.

ANSI and Unicode

NT is a Unicode OS and Win9x are ANSI. Don't use Unicode on Win9x applications. If ANSI is used on NT a wrapper translates everything to Unicode, slowing the system.

To use Unicode under WinNT a UNICODE symbol has to be defined and the _T macro must be used before every string. Also appropriate char types must be used.

(See also UNICODE programming summary)

Screen Coordinates

Under Windows 9x screen coordinates are limited to 16 bit, staying in the range –32768 to 32767. Under Windows NT/2000 screen coordinates are 32 bit. (See also CDC)

Platform SDK vs. MFC

Use Platform SDK only for very small applications that use very few common controls or if you need to use only functionality not supported by the MFC framework.

Use MFC in all the other cases. If something is possible with MFC use it and do the rest with Platform SDK.

MFC Regular vs. MFC Extension DLLs

MFC regular DLLs can be used in every Win32 programming environment because only C functions are exported. C++ classes can be used inside the DLLs.

MFC extension DLLs can be used only with C++ compatible compilers, because MFC derived classes, member functions, C++ classes and so on can be exported.

(See also Extension DLLs)

Message Routing

MFC objects can receive messages if they have a message map. A message map contains message IDs and pointers to handlers. When the object receives a message, it looks at its own message map for a matching message. If the message is found the handler is executed, otherwise the message is routed to the object at the lower level.

If multiple handlers exist, the higher is called. (See also Message Handling and Mapping Topics)

The handling order is the following:

SDI	MDI
View	View
Document	Document
	Child Frame
Main Frame	Main Frame
Application	Application

Document/View Architecture

In Document/View architecture, an object manages data and another takes care of presentation, allowing changes to how the data are showed without changing how the data are managed.

Single Document Interface (SDI), Multiple Document Interface (MDI) and Dialog Based are the three possible choices. In the SDI architecture there can be only one document opened.

(See also Document/View Architecture Topics)

MFC Drawing architecture

To draw on the screen, Windows applications have to deal with GDI and device contexts (DC).

MFC supply a class, CDC, which encapsulates the device context. With CDC it's possible to draw lines, ellipses and so on. The most important methods are MoveTo, LineTo, PolyLine, PolyLineTo, Arc, ArcTo, PolyBezier, PolyBezierTo, PolyDraw, Chord, Ellipse, Pie, Polygon, Rectangle, RoundRect.

MFC also supplies a set of graphic components like Pens, Brushes, Palettes, Bitmaps, and Fonts.

To draw a text with MFC it's necessary to set the attributes of the text before writing it.

SetTextColor, GetTextColor, SetBkMode, GetBkMode, SetBkColor, GetBkColor, SetTextAlign and GetTextAlign are used to set the attributes. TextOut, TabbedTextOut, DrawText and ExtTextOut are used to write the text on the device.

GDI translates logical coordinates to physical coordinates using several mapping methods. There are 8 mapping modes, MM_ISOTROPIC, MM_ANISOTROPIC, MM_HIENGLISH, MM_LOENGLISH, MM_HIMETRIC, MM_LOMETRIC, MM_TEXT and MM_TWIPS.

MFC Printing and print-preview architecture

MFC functions useful for printing support are embedded in all the classes derived from CView and are OnPreparePrinting, DoPreparePrinting, OnBeginPrinting, OnEndPrinting, OnEndPrintPreview, OnPrepareDC, OnDraw and the most important, OnPrint.

Overriding OnPrint allows providing special printing functions, like header, footer and so on.

(See also Printing and Print Preview Topics)

Multithreading

Interface threads

Can receive and process messages

Worker threads

Cannot receive messages. A worker thread is only another path of execution for the application to do background work.

(See also Multithreading Topics)

Process priorities

- REALTIME_PRIORITY_CLASS — Higher
- HIGH_PRIORITY_CLASS
- NORMAL_PRIORITY_CLASS — Default
- IDLE_PRIORITY_CLASS — Lower

An application can change its priority using SetPriorityClass()/GetPriorityClass()

Thread priorities

- THREAD_PRIORITY_TIME_CRITICAL
- THREAD_PRIORITY_TIME_HIGHEST
- THREAD_PRIORITY_ABOVE_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_LOWES
- THREAD_PRIORITY_IDLE

An application can change the priority of an individual thread using SetThreadPriority() / GetThreadPriority()

Synchronization

CCriticalSection

Only a section of code can be accessed at one time

CEvent

Can block a process from accessing a resource until another thread allows it

CMutex

Used to lock a resource shared by multiple threads

CSemaphore

Allows a limited number of accesses to a resource

CSingleLock

Used to control access to previous synchronization objects

CMultiLock

Can block up to 64 synchronization objects

Database

ODBC

ODBC was the standard to access a Relational DB. ODBC is still most popular, but Microsoft prefers that new applications use OLE DB to access a DB.

RDO is the object model built over ODBC.

OLE DB

OLE DB is the new set of APIs to access Relational and non-Relational data. The most important benefit of using OLE DB is that the same model is used to access every kind of data (with the right provider). An ODBC provider for OLE DB is given for backward compatibility with every relational database.

ADO is the object model built over OLE DB.

JET

JET is the database engine used by Microsoft Access and is very popular for desktop databases.

DAO is the object model built over JET.

Access methods

MFC

MFC can directly access ODBC or DAO databases. (See also Database Topics (General))

ATL

ATL can access OLE DB databases.

Platform SDK

The Platform SDK is the hardest way to access a database, and can be used if the other two libraries are not sufficient to do what the application needs.

ADO and RDO object models can be accessed only by importing the type libraries and by directly using COM objects.

(See also Choosing an API to see differences between DB access technologies)

Designing Properties, Methods and Events of ActiveX Controls

ActiveX controls in MFC are implemented with the COleControl class.

Properties

There are 9 stock properties common to every ActiveX control already implemented in a control that derives from COleControl base class:

- Appearance
- BackColor
- BorderStyle
- Caption
- Enabled
- Font
- ForeColor
- hWnd
- Text

There can also be custom properties that can be implemented in four ways:

- Member variable
- Member variable with notification
- Get/Set Methods
- Parameterized

Methods

There are two stock methods implemented by COleControl class:

- DoClick
- Refresh

COleControl class doesn't support custom methods. To add a custom methods a programmer must use the DISP_FUNCTION() macro, and add the ID of the statement in the primary dispatch interface in the .ODL file.

Events

These are the stock events implemented by COleControl class:

- Click
- DblClick
- Error
- KeyDown
- KeyPress
- KeyUp
- MouseDown
- MouseMove
- MouseUp
- ReadyStateChange

Like custom methods, custom events are also not supported by COleControl class.

Custom events can be implemented with the EVENT_CUSTOM macro. Also the .ODL file must be modified to add the ID statement in the primary dispatch interface.

Establishing the Development Environment

Visual C++ Installation

Requirement	Standard Edition	Professional Edition	Enterprise Edition
Operating system	Windows 95 or later, Windows NT 4.0 (SP 3 or later), Windows 2000		
Processor	486/66 minimum Pentium 90+ recommended	Pentium minimum Pentium 90+ recommended	
RAM	24 MB minimum, 32 MB recommended		
Free Disk Space	Typical: 225 MB Full: 305 MB	Typical: 290 MB Full: 275 MB	Typical: 305 MB Full: 405 MB
Other	CD-ROM, mouse, VGA monitor (SVGA recommended), Internet Explorer 4.01 SP 1		
Additional products			
Internet Explorer	Typical: 43 MB, Full: 59 MB		
MSDN	Typical: 57 MB, Full: 493 MB		
Windows NT Option Pack	Not included		WinNT: 200 MB Win9x: 20 MB
SQL Server 6.5	Not included		WinNT: 80 MB typical, 95 MB full
SNA Server 4.0	Not included		WinNT: 50 MB typical, 100+MB full

Click to see a Comparison Chart of Features in Each Visual C++ Edition

Visual SourceSafe

Visual SourceSafe is a source code control system that comes with Visual C++ and Visual Studio Enterprise Edition. (See also VSS Start Page)

VSS Client is installed on every developer's machine, allowing access to the centralized database installed and administered with VSS Server.

NetSetup is the best way to install VSS over a network. It installs only VSS Client, not VSS Server.

It is stored on the same directory on the server where the VSS Server is installed, and can be accessed over the network. No CD is necessary.

With NetSetup users can install VSS Client without the help of an Administrator.

After the installation of VSS Server, administrators have to configure it with VSS Administrator.

If the VSS Server directory is shared it's better to set the right read-write permissions to users.

If project security is enabled, a user can have four levels of access:

- read-only
- check out/check in
- add/rename/delete
- destroy

If security is not enabled there are only read-only and read-write permissions. VSS Administrator is used to set access rights.

VSS Client allows the user to store and retrieve files and to ensure that only one person at a time can modify a file. Normally only one user can check out a file at a time, but this can be changed.

Files can be shared among multiple projects. Changes made to a file are seen by all the projects.

With branching a file can go in two directions. Under the Paths' tab it's possible to see the history of a branched file.

Get Last Version is the command used to retrieve the last version of a file. Also an older version can be retrieved from History of File/Project.

A file with Get Last Version can be retrieved only if it doesn't exist, or is in read-only state. If the file is not in read-only mode, VSS assumes that the file is checked out and doesn't replace it.

Label is the command used to mark all the files in the project. It's useful to mark all the files before a major release so that if the release needs to be restored files can be immediately found.

Implementing the Navigation for the User Interface

MFC AppWizard

Under the MFC AppWizard three kinds of projects can be created: Single Document Interface, Multi Document Interface and Dialog Based.

It's possible to enable or disable the support for the Document/View architecture.

Database support	
None	
Header Files Only	It adds only the file AFXDB.H and links all the libraries
Database View without File Support	It gives the ability to use the CRecordView derived view class but does not give any serialization option
Database View with File Support	It adds document serialization

Active Document Support	
None	
Container	It allows the inclusion of Active Documents generated by other servers
Mini Server	It allows the creation of documents that can be embedded in other applications
Full Server	It allows the Active Documents to run as a stand-alone application
Both container and server	

Other options can be set like the support for ActiveX, Automation, Status Bar, Docking Toolbar, 3D Controls, MAPI, Winsock, Printing support and Context Sensitive Help.

Also some advanced options can be set like the name of the classes, the styles of the windows, and so on.

Resource Editor

The Resource Editor can be used to work with .RC files. There are editors for Toolbars, Menus, Dialogs, Strings, Accelerator Keys, HTML Resources, Graphics and Binary Files.

To add an accelerator key directly in the Dialog Editor, or in the Menu Editor just add "&" before the letter ("Save &As" for example)

(See also Resource Editor Topics (Specific to Visual C++))

Toolbars with MFC

Toolbars can be created with the Toolbar Resource Editor. It's possible to create new toolbars, buttons and separators, convert bitmaps into resources and edit existing toolbars or buttons.

The "Prompt" edit box in the "Toolbar Button Properties" allows you to set status bar text and tool tip. For example, with the text "Opens an existing Document\nOpen", when the cursor is over the button, "Opens an existing Document" is displayed in the status bar and the tool tip "Open" is showed under the button.

To add a custom toolbar to a window a CToolBar protected member has to be added, and, in the OnCreate function, the CreateEx function can be called. To enable tool tips the CBRS_TOOLTIPS style has to be passed to the function.

(See also Toolbar Topics)

Status bar with MFC

To write in a status bar there are different ways: CStatusBar::SetText() and CCmdUI::SetText()

(See also Status Bar Topics)

Class Wizard

Class Wizard can be used to create new classes, add member variables to existing classes, and manage message maps and message handlers.

Class Wizard can also used to associate a Resource to a Class. If it's possible to map the resource to a class the line Class Wizard will be in the context-menu of the resource.

Property Sheet

A property sheet is a tabbed dialog box. To create a property sheet a programmer must create a set of dialogs with the resource editor and has to map them to a set of classes derived from CPropertyPage.

Then a class derived from CPropertySheet must be created. All the property page members must be added in the .h of the property sheet class with statements like `CPropPage1 m_PropPage1`.

In the constructor every property page must be added to the sheet with `AddPage(&m_propPageX)` for every page. AddPage() is member of CPropertySheet.

Now creating the property sheet and calling the DoModal will display the tabbed dialog box.

(See also Property Sheet Topics)

CFormView

CFormView is used to show controls, usually used in dialog boxes, into normal Views.

(See also CFormView)

Process and Validate User Input

DDV or Dialog Data Validation is the mechanism used to provide simple validation to data using CString or numeric data types. It can be set by the ClassWizard or directly with DDV_MaxChars, DDV_MinMaxInt or with all the other macros.

DDX or Dialog Data Exchange is the mechanism used to transfer data from a document's variables to a dialog box and vice versa.

DoDataExchange is the function that calls DDX and DDV macros.

First a DDX macro maps a control to a variable, and then a DDV macro defines the rules that apply to the variable. Every DDV macro refers to the previous DDX macro.

(See also Dialog Data Exchange and Validation)

UpdateData(FALSE) forces the data to be transferred from the document to the dialog; UpdateData(TRUE) force the opposite direction. If some of the rules are violated UpdateData returns FALSE.

ActiveX Controls

With the AppWizard setting "ActiveX support" enabled, it's possible to use ActiveX controls directly in a project, using them like every other control.

To add an ActiveX control to a project it's necessary to use Add to Project/Components and Controls from the Project menu. There it's possible to select an ActiveX control and use it.

A wrapper class will be created and all the functions of the control become accessible.

ISAPI DLLs

ISAPI are extensions of the Internet Information Server that can extend its functionality.

ISAPI Server extensions are equivalent to the CGI. They are used to generate dynamic content in the Web Site in response of a request of the ISAPI DLL.

ISAPI Filters can pre and post process all the data sent from and to the Web Browser.

The AppWizard can create ISAPI DLLs by selecting the appropriate items.

ISAPI Server related classes are CHttpServer and CHttpServerContext.

ISAPI Filter related classes are CHttpFilter and CHttpFilterContext**.**

(See also Internet Server API (ISAPI) Extensions)

Scriptlets

Scriptlets cannot be used directly in this version of Visual C++. The only way to use them is to embed them in an HTML pages and to show the HTML page with CHtmlView or with the Internet Explorer ActiveX.

Store and Retrieve settings from the registry

With the support of MFC, settings can be stored under HKEY_CURRENT_USER\Software*CompanyName*\AppName*SectionName*

The SetRegistryKey() member function of CWinApp derived classes is used to set the *CompanyName*.

AppName is the same used for the Application in project settings.

SetProfileString(), GetProfileString(), SetProfileInt() and GetProfileInt() can be used to store and retrieve string and integer data from the registry. *SectionName* is the first parameter of these functions.

RegisterShellFileTypes() allows saving all the information about document types and associations related to the application.

To write under other registry keys, Win32 APIs are needed. The most common are RegCreateKeyEx, RegOpenKeyEx, RegCloseKey, RegDeleteKey, RegSetValueEx, RegQueryValueEx and RegDeleteValue.

Display data from a data source

Serialization is the process of allowing objects to persist between runs of your program

An object must derive from CObject, must use the macros DECLARE_SERIAL and IMPLEMENT_SERIAL and must implement **Serialize** method. Serialize receives a CArchive in which the data have to be stored.

CArchive is the class that stays in the middle between the Serialize method and the CFile. CArchive allows moving data in only one direction once they're created. To move data it's possible to use "<<" or ">>".

CFile is the class that allows you to write and to read a file. CStdioFile is derived from CFile and allows access to text files.

CSocket and CAsyncSocket can be used to use WinSock to send data across a network.

CSocket is simpler and allows the programmer to create client/server applications, which communicate with each other using sockets. CArchive is needed to manage the communication process.

Displaying data from a database to the screen (using CRecordSet to access the database) is available via the class CRecordView. A dialog template (shown inside the CRecordView) contains controls that can be mapped to the fields of the database.

Instantiate and Invoke a COM component

To instantiate and invoke a COM component under Visual C++ there are two main ways.

The smartest way is to import the DLL of the component and to let the environment take care of all the implementation details.

This is done in this way:

```
#import "file name with path.dll" no_namespace
```

Using no_namespace tells the compiler to not add a namespace to the component. If there are many components with the same function names, it's better to use namespaces.

If the component has an interface called IFoo, to use it a programmer can do:

```
IFooPtr pFoo(_uuidof(Foo));
pFoo->DoWhatYouWant();
```

Smart pointers are used to call AddRef, Release, QueryInterface and so on.
(See also The #import Directive)

The other, classic, way is to initialise COM with AfxOleInit() or CoInitializeEx(), identify the ClsID of the component using ::CLSIDFromProgID(), and call CoCreateInstance to create the component.

Then Release has to be called when the component is not needed, and CoUninitialize() has to be called at the end of the program.

Asynchronous Processing

Threads

Secondary threads can be created to do background tasks or to interact with the user.

The function has to follow the prototype `UINT ThreadName(LPVOID paramName)` and can be started with `AfxBeginThread()`.

Here is an example:

```
UINT foo(LPVOID lpFoo)
{      ……     }
CWinThread* pFoo= AfxBeginThread(foo, NULL);
```

Download ActiveX user interface controls

To download an ActiveX control from a Web page the OBJECT tag is used, with the CLASSID tag that specifies the ClsID of the control and the CODEBASE tag that specifies the location of the control. With CODEBASE it is also possible to specify the version of the control, so if a new version exists, the browser will not use the control installed in the client machine, but will download the new version.

The control must be supplied in a .CAB file that contains the .OCX file and the .INF file that specifies how to install the control. The .CAB file can also be signed to ensure who is the maker of the control.

Implement online user assistance

Writing relevant information for the user in the status bar is the first method to provide assistance.

To write in the status bar the best way is to use SetText(), like

```
m_stBar.SetText("Ok")
```

Tool tips are supplied by MFC for toolbar buttons and menus, but can be added to every control.

This can be obtained with the help of CToolTipCtrl:

```
CToolTipCtrl* pToolTip;
pToolTip->Create(pDialogWnd);              // pointer to the window
pToolTip->AddTool(pControl,"Tooltip…")     // pointer to the control
```

On line Help and Context Sensitive Help are implemented by the framework (if selected in the MFC AppWizard).

The WinHelp() member of CWinApp is used to call the standard Windows Help.

To use HTML Help it's necessary to use run "hh.exe" because at the moment there is no support from the compiler. (See also HTML Help Start Page)

To link the help file to a compiled HTML help (also on a Web Site) it's possible to add a macro in the source RTF file (See also To link from a topic in a WinHelp file to a topic in an HTML Help file):

```
!execfile(hh.exe, ms-its:file name.chm::/topic.htm)
```

Error Handling

Exceptions are objects that contain error conditions. Exceptions are produced by functions when there are errors.

With MFC there are two ways of catching errors: MFC macros and C++ exceptions.

MFC macros are only for backwards compatibility. (See also Exception Handling Topics (General))

CException is the base class of every MFC exception. It has two methods to report errors, GetErrorMessage() that retrieves the error message and ReportError() that retrieves it and reports it to the user.

```
try
{// piece of code that could generate exceptions
}
catch (CMemoryException* memExc)
{     // first catch a memory exception
      memExc->Delete();
}
catch (CFileException* filExc)
{     // then a file exception, it's just a sample
      filExc->Delete();
}
catch (CException* allExc)
```

```
{     // this catches all the other exceptions…
      allExc->Delete();
}
```

Remember to use Delete() method of the exception to delete it, because it's not sure that the exception is on the heap or on the stack.

CException catches all the exceptions, so it must be the last in the order, because C++ exceptions are handled in the order they are declared.

An exception can also be ignored, and it will be catch by the next handler. An unhandled exception can cause the termination of the program.

Use an Active Document

Active Documents are stand-alone applications hosted by other applications (like MS Office or Internet Explorer).

Using Active Documents in VC applications is very simple. By using MFC AppWizard it's possible to select the type of support needed.

Creating and Managing COM components

Create a COM component

SDK

Using the SDK is the hardest way to create a COM component. Everything must be programmed, the registration of the component, the specifications of the interfaces in .IDL, the ClassFactory, and the class that implements the component.

MFC

CCmdTarget is the class that implements IUnknown and IDispatch. MFC is very heavy and should be used to make COM components if it's used very intensively. With the Wizard only OLE Automation server and clients can be easily built.

ATL

ATL is the best choice to implement a lightweight COM component. Using the Wizard is the fastest way to make a COM component with ATL.

Create ActiveX user interface controls

ATL

ATL generated controls are very light. To create a control the "ATL COM AppWizard" should be used. After that the programmer should use "New ATL Object", "Full control".

In the "ATL Object Wizard Properties/Attributes" the programmer can choose the threading model, the ability to support aggregation, dual interfaces, connection points, and ISupportErrorInfo for the FreeThreaded Marshaler.

SDK

To implement a control without ATL or MFC a programmer must implement IOleControl, IOleControlSite and ISimpleFrameSite. After that the control must support OLEIVERB_PROPERTIES and events. The control has to draw itself in the container space.

MFC

Use the "MFC ActiveX Control Wizard" to create the control. The control is based on COleControl that is based on COleControlModule. MFC ActiveX controls are very heavy, and should be used only if MFC is used extensively.

(See also ActiveX Controls: Overview)

Reuse Existing Components

(See also Object Reusability)

Containment

A component is "contained" inside another component. When a request for an interface of the "contained" component is requested, the call is sent to an interface of the container that calls the inner interface. (See also Containment)

Aggregation

With aggregation the inner interface is directly exposed to the client without a wrapper. The inner component must support aggregation, because for every QueryInterface done on the inner component it has to check its interfaces and the interfaces of the outer component. (See also Aggregation)

Error Handling

IErrorInfo

Is supplied with the error object by the OS (CreateErrorInfo API)

ICreateErrorInfo

Is supplied with the error object by the OS (CreateErrorInfo API)

ISupportErrorInfo

Is used by the Automation server to report errors to the client

(See also Error Handling Interfaces)

Log errors in an error log

Under Windows NT/2000 the application error log can be used to store error messages of every purpose. Use EventViewer to read the log. Use **ReportEvent()** Win32 API to write in the application error log.

Create and use an Active Document

Active documents are implemented with additional interfaces that manage views, so that objects can function within containers and yet retain control over their display and printing functions.

COleServerDoc is the replacement of CDocument that supports Active Document creation.

IOleObject, IOleClientSite, IOleDocumentView, IOleCommandTarget, Iprint, IDataObject, IPersistStorage, IOleInPlaceActiveObject, IOleInPlaceObject, IPersistFile are interfaces needed on the server.

IOleInPlaceSite and IOleInPlaceFrame are used on the container.

(See also Active Documents)

Debug a COM component

To debug a COM component the easiest way is to set a breakpoint into the component, build it in debug mode, and register the component.

Then a programmer can load the client into the IDE and start it in debug mode. When the client calls the component, the breakpoint stops the execution and transfers it to the debugger.

ATL based COM components can also be debugged by using special macros.

_ATL_DEBUG_INTERFACES is used to enable reference count debugging.

_ATL_DEBUG_QI is used to enable QueryInterface debugging.

Apartment-Model Threading

Single-Threaded Apartment

COM calls are done by sending messages to the window's message queue. This technique allows synchronizing concurrent calls to be serial. Legacy code often ignores threading, and by default uses a single STA. (See also Single-threaded apartments)

To enter the STA the thread must call
`CoInitializeEx(NULL,COINIT_APARTMENTTHREADED)`

Multithreaded Apartment

COM calls are sent directly to the object. The object has to deal with synchronization, because multiple calls could arrive from multiple threads. (See also Multi-threaded apartments

To enter the MTA the thread must call
`CoInitializeEx(NULL,COINIT_MULTITHREADED)`

Creating Data Services

Accessing and manipulating data by using ad hoc queries

ODBC

To use ODBC in a Visual C++ application two MFC classes are necessary.

CRecordset is used to interact with rows returned from the database in response of a query. CRecordset is never used directly; it's better to use a derived class.

CDatabase is used to attach and to communicate with a database and is generally used when there are more recordsets.

(See also ODBC and MFC)

ADO

ADO is the object model built on the top of OLE DB. Remember to use ADO or OLE DB in every new application. To use ADO with Visual C++ a programmer needs to import the type library "`msado15.dll`".

The object model exposed by ADO is composed of six main objects, but not all are necessaries to query the database. For example a recordset can be obtained without opening a connection and without sending a command, but using a connection allows you to obtain more than one recordset, and using a command allows you to send the same command without querying the metadata every time.

Connection

Used to maintain connection information like cursor type, connection string, time-outs, default database

Error

Used to report extended error information. A collection of errors is used because one or more errors could be returned.

Command

Contains information about a command, like the query string, parameters, and so on.

Parameter

The Command object can contain a collection of parameters. Each parameter type can be declared from the programmer to improve performance, or can be discovered at run-time.

Recordset

Is a set of rows returned from a query, including cursors.

Field

Is used to contain a set of information about a single column of data.

(See also Using ADO with Microsoft Visual C++)

DAO

DAO is the object model used to access the Jet database engine. It's supported by MFC with these classes: CDaoWorkspace, CDaoDatabase, CDaoException, CDaoQueryDef, CDaoRecordset, CdaoFieldExchange

Remember that CDaoRecordset is never used directly; a derived class is used instead.

(See also DAO and MFC)

RDO

RDO is an object model built over ODBC. It's not supported by MFC, but can be used by importing the type library and accessing the COM objects directly.

Handle database errors

The ADO Connection object contains an error collection with these methods and properties

Count

Contains the number of error in the collection

Item

Is used to retrieve an error from the collection

Clear

Is used to remove all the errors from the collection

The Error object contains these properties: Description, Number, HelpFile, HelpContext, Source, SQLState, NativeError

(See also Errors Collection (ADO))

Testing and Debugging the Solution

Debugging Techniques

Debugging Support

Visual C++ includes debugging support in the run-time library, enabled in the debug version of the executable.

There are debug versions of `malloc`, `free`, `calloc`, `realloc`, `new` and `delete` which are useful to find memory leaks.

(See also Using C Run-Time Library Debugging Support)

Also in the IDE of the compiler there are many debug features, like the integrated debugger with advanced breakpoints, edit-and-continue, etc…

(See also VC Debugger)

Depends

Depends (and other tools like QuickView and DumpBin) can be used to find the dependencies of and executable to discover which dll, ocx or com components are used by an application and which are missing or with a wrong version.

Spy++

Spy++ can be used to inspect windows, processes, threads and messages.

(See also Spy++)

MFC Macros

ASSERT

ASSERT allows the programmer to check for logic errors during the execution of a program. If the condition is false the program stops and displays an error message. This can be used to check for pre and post-conditions. ASSERT works only in debug mode, cleaning the release mode of all the testing code.

ASSERT_VALID

ASSERT_VALID is used with objects derived from CObject. It's the same of ASSERT and it also calls the AssertValid function of the object.

ASSERT_KINDOF

ASSERT_KINDOF checks if the object is a member of the specified class.

TRACE

TRACE is used to send strings on a dump device. Use TraceR.EXE to enable tracing.

DEBUG_NEW

DEBUG_NEW is useful to find memory leaks. To use DEBUG_NEW the macro

```
#define new DEBUG_NEW
```

must be used. DEBUG_NEW logs every memory allocations in a file. CMemoryState::DumpAllObjectsSince can be used to view all objects allocated.

Elements of a Test Plan

Beta test

A beta test is the testing made by users that are not part of the development process. A beta test is useful to test the application with many different conditions like OS, CPU, RAM, drivers, user level and different languages.

Regression test

Regression test is repeating the same test to check that nothing is changed from the last version of the application in areas not affected by the development process.

Unit test

Unit test is the testing made by the developer on a small working part of an application. Unit test can imply writing stubs to simulate other units not yet written. Use only to try if a unit works.

Integration test

Integration test is used to check if all the units can work together. Unit testing is not enough to show if different units can work together.

Stress test

Stress test is needed because an application can stop working if there is too much load. Stress testing places the highest loads with the lowest amount of resources available. This kind of test is useful to determine the minimum requirements and the maximum load for an application.

Deploying an Application

Creating a Setup program

Use QuickView, Depends or DUMPBIN to determine dependencies.

Use InstallShield to create the setup program.

Remember to include all DLLs, COM and ActiveX components that are used by the application.

Setup created with InstallShield will register all COM components into the registry.

Self-registering DLLs must contain DllRegisterServer and DllUnregisterServer functions. These functions are called to register the COM component into the registry.

Self-registering EXEs must accept \RegServer and \UnregServer command line options. These are used to register/unregister the component without running it.

(See also Redistributing Microsoft Visual C++ 6.0 Applications)

Setup program must be called SETUP.EXE to comply with the Windows 98 Logo Program

Using .cab Files

Cabinet or .cab are compressed files, used to distribute applications.

A cab file can be signed, and can be deployed over a network.

To sign a cab file a digital certificate is required.

A cab file can be embedded in an HTML page (using the tag OBJECT and CODEBASE)

A cab file can contain an .INF file to specify which components must be registered.

Plan Floppy Disk, Web and Network Deployment.

InstallShield allows selecting the media used to distribute the application.

An application can be distributed over a set of floppies or compact disc, over a network or using a Web-Based distribution. Compact disc distribution can also be made automatic by providing an AUTORUN.INF file on the CD. A simple file is:

- [AutoRun]
- open=filename.exe
- icon=filename.ico

- where open is the file to run and icon is the icon of the CD.
- Compact disc is the default method for InstallShield.

Evaluating Microsoft SMS

Microsoft System Management Server can be used to deploy applications over a network.

A package is the basic unit of software distribution under SMS.

The SMS installer can be used to create a Package Definition File that can be distributed and launched on every client to install the package.

(See also Advanced Desktop Management: Systems Management Server)

Uninstaller

InstallShield automatically generates a program called UNINST.EXE that reads installations log files (UNINST.ISU) and removes any items installed. UNINST.EXE doesn't remove hidden items except .GID, .FTG and .FTS files. UNINST.EXE also doesn't remove items added after the installation of the application, including user's documents.

To register an Uninstaller (with a custom setup program), an entry must be added to the registry key under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CurrentVersion\Uninstall\AppName, containing the path of the Uninstaller program.

(See also part 2.5 Support Add/Remove Programs properly of the Application Specification for Microsoft Windows 2000 for Desktop Applications)

(See also Removing "Ghosts" from the Add/Remove Programs Utility)

Zero Administration for Windows (ZAW)

Zero Administration for Windows is an initiative to reduce work and costs associated with installing and managing a Windows-based network environment.

ZAW includes functionalities to automatically update systems and install applications, to cache configuration information and to administer and lock systems from a central point.

Windows Installer and Systems Management Server are part of the ZAW initiative.

To implement a ZAW solution the Zero Administration Kit is needed.

It can run on Windows NT and Windows 9x.

An evolution of the ZAW is available and is an important part of Windows 2000.

Maintaining and Supporting an Application

Fix errors and prevent future errors

Logic errors (known also as bugs) occur when the program can be compiled, but doesn't work as expected.

Syntax errors occur when the compiler doesn't understand what is written in the source code.

TRACE, ASSERT, SEH (Structured Exception Handling) and C++ exceptions can be used to prevent future errors and to trap run-time errors when they occur.

Deploy updates

Use InstallShield to deploy application updates. It can be used to replace program files, components and registry settings. Use the Overwrite property to specify the conditions under which files will or will not overwrite the user's hard disk.

Special thanks to

Lorenzo Barbieri

for contributing this Cramsession.