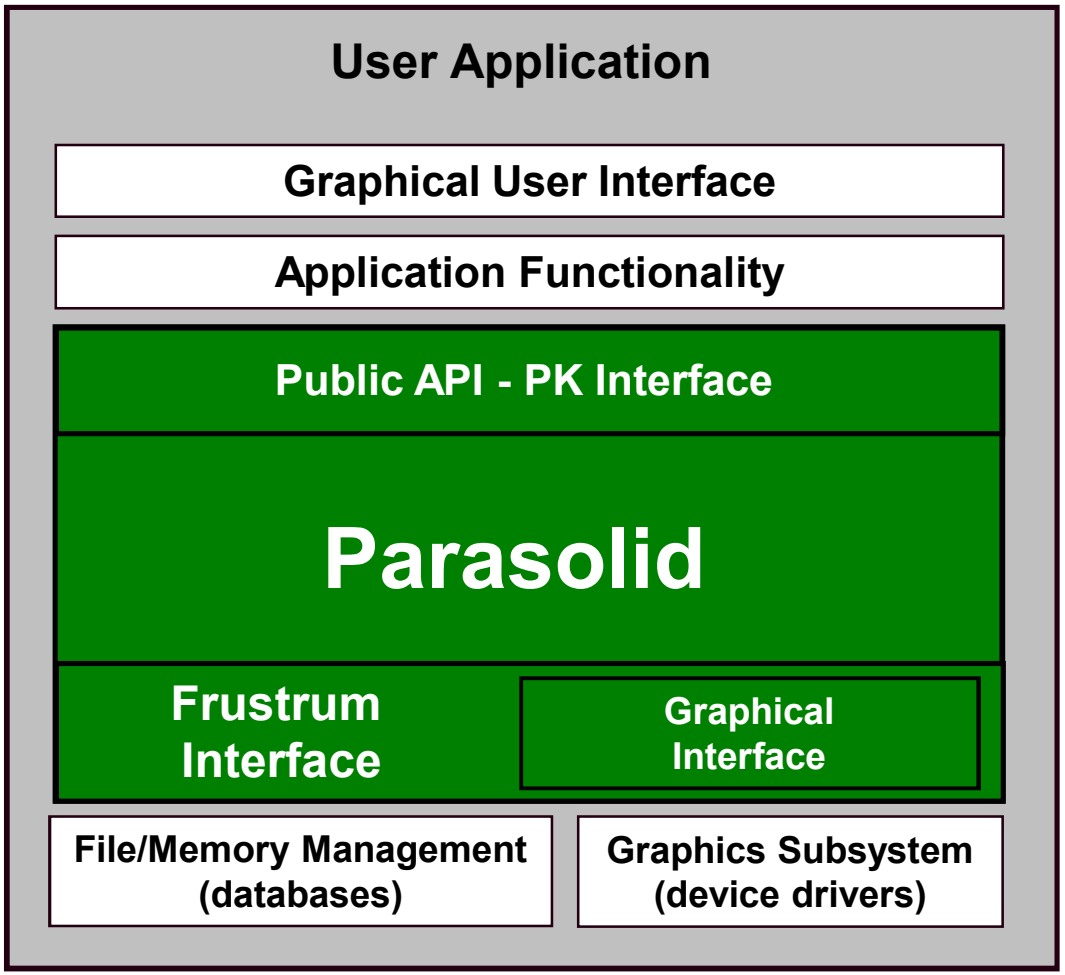
Parasolid Kernel 2

Human Centered CAD Laboratory

Interfaces to Parasolid



- Parasolid is accessed via the PK interface to build and manipulate objects
- Parasolid communicates with the operating system via the frustrum and the graphical interfaces

Interfaces to Parasolid

- Downward interfaces
  - Your application controls all interaction between Parasolid and the operating system. The parts of your application that deal with this are referred to generically as downward interfaces. Because Parasolid itself makes calls to routines in these downward interfaces, your application must supply them and they need to be registered with Parasolid before your application can call any Parasolid functions.
- Two parts to downward interface:
  - The frustrum and Graphical Output (GO)
- The frustrum interface
  - The frustrum is a suite of functions that deals with:
    - File handling: Saving and retrieving Parasolid part files and other data
    - Memory management: Allocating and freeingn memory for internal calculations and data structure storage

Interfaces to Parasolid

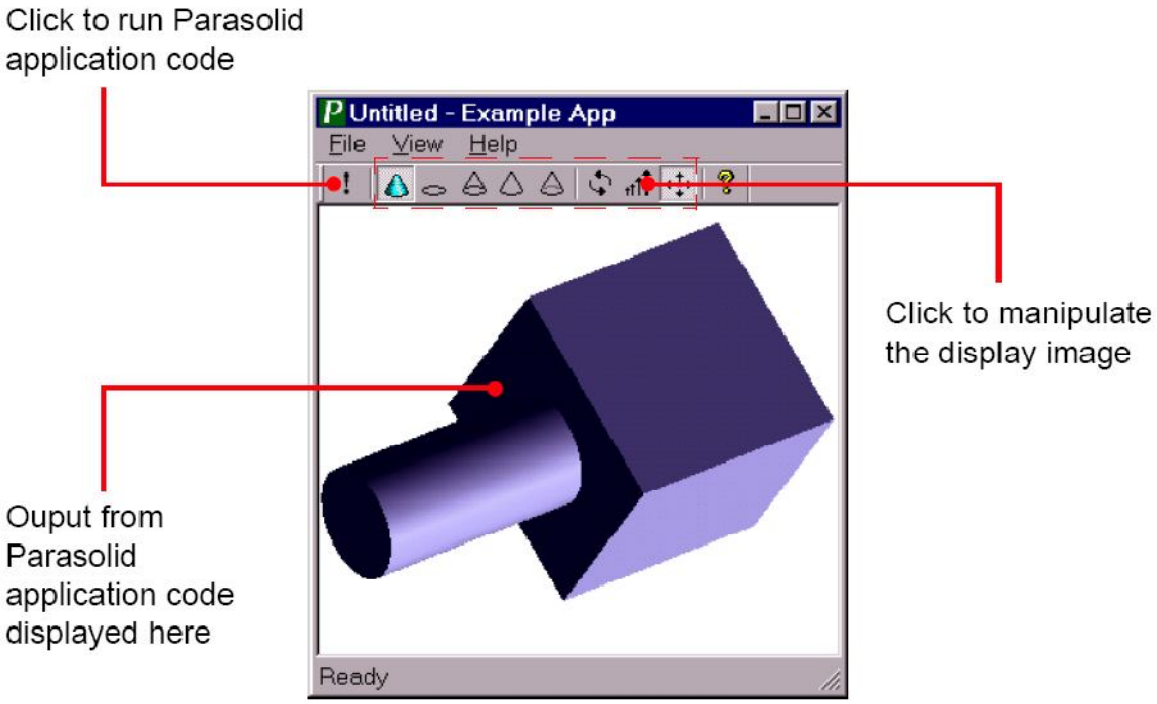
- Graphical output interface
  - You must also provide GO functions for Parasolid so that they could be called to display models– whether by rendering them on screen, or printing them to a plotter or laser printer.
  - Parasolid calls the GO functions whenever your application calls a Parasolid rendering function in order to draw one or more parts. The GO functions encapsulate the graphical information output by Parasolid, and pass it to a graphics library (that you also provide) in order to render the image.
  - You can write the graphics library yourself, or you can use a third party library such as OpenGL or DirectX

Example Application

- Development Environment
  - Microsoft Visual Studio .NET 2003 (C++ .NET)
  - Parasolid Kernel V16 (or later)
  - OpenGL Graphics Library
- Loading the project into MSVC++
  - Open the file Source\Example App.sln
- This solution is INCOMPLETE
  - But you will make it COMPLETE

Example Application

- Result of Example App
  - The Example Application has a single window that contains a menu bar, toolbar, and a viewing area that is initially empty.



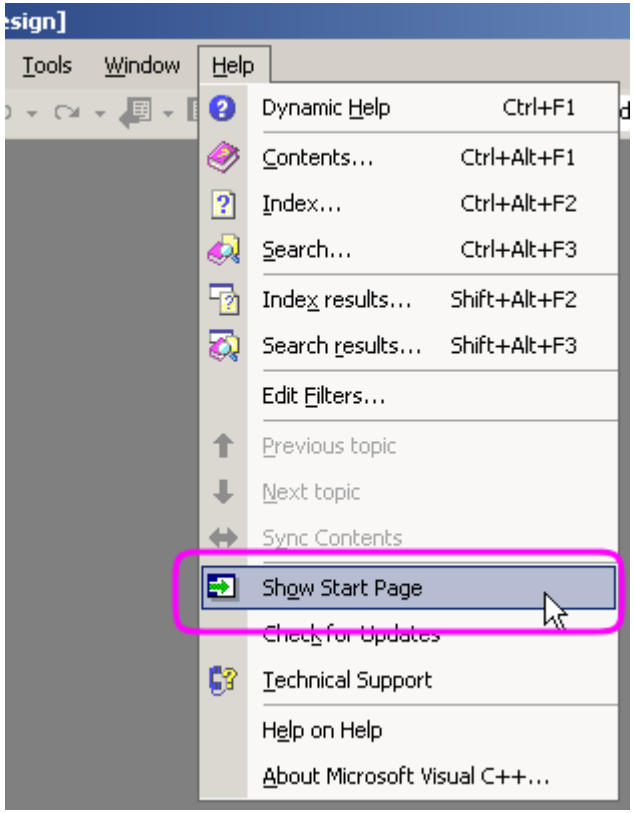
Example Application

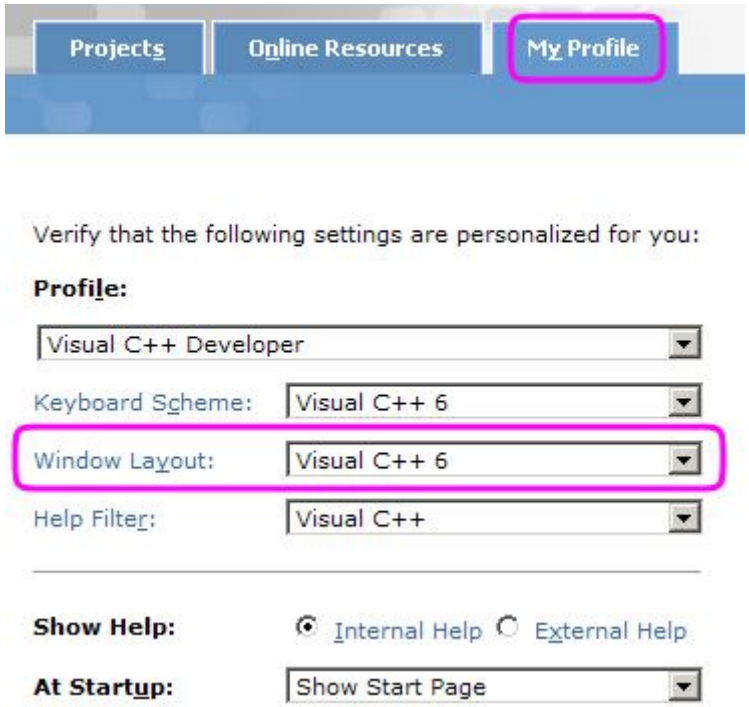
- Buttons in the toolbar
  - The tool bar in the example application contains a number of buttons, as described below:

Click this button …	To perform this action …
	Run the Parasolid application code that you want to prototype. This calls CMyCode::RunMyCode and then displays the output from the calls to Parasolid functions as follows: ■ If the code is written as a series of steps (using a case statement) a single step is completed and the output at that point displayed. To perform the next step, click again. ■ If the code is written as one large fragment, the parts present when the code has completed are displayed.
	Display the current state of the bodies in the Parasolid session in shaded, wireframe, wireframe with silhouettes, hidden line, and gray hidden line modes respectively. **Note:** Orphan geometry is only displayed in wireframe mode.
	Rotate, zoom into or out from, or pan the view respectively, using the mouse button.
	Display version information about the Example Application.

Window Layout Setting

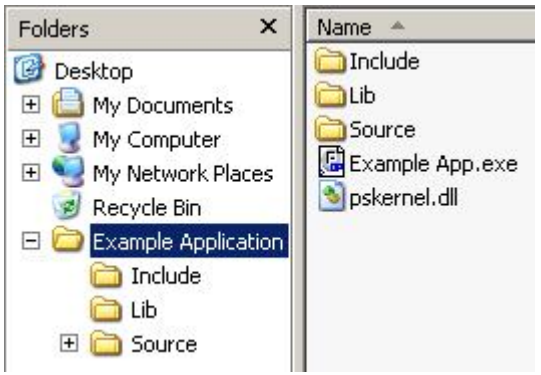
- Menu > Help > Show Start Page > My Profile

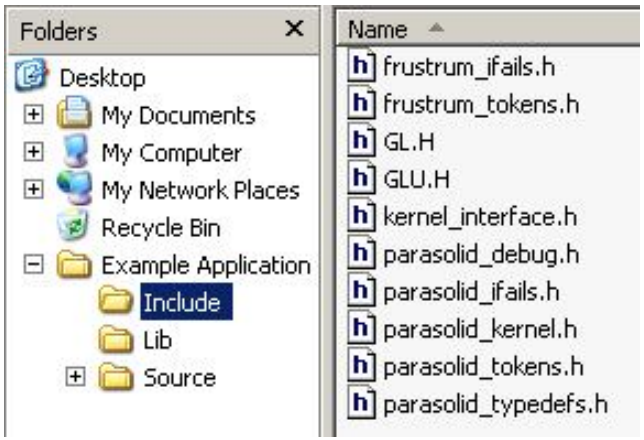


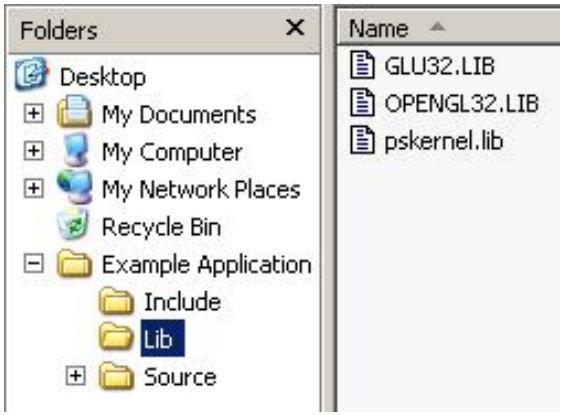


Parasolid Files

- Include/*.h files
- Lib/*.lib files
- pskernel.dll







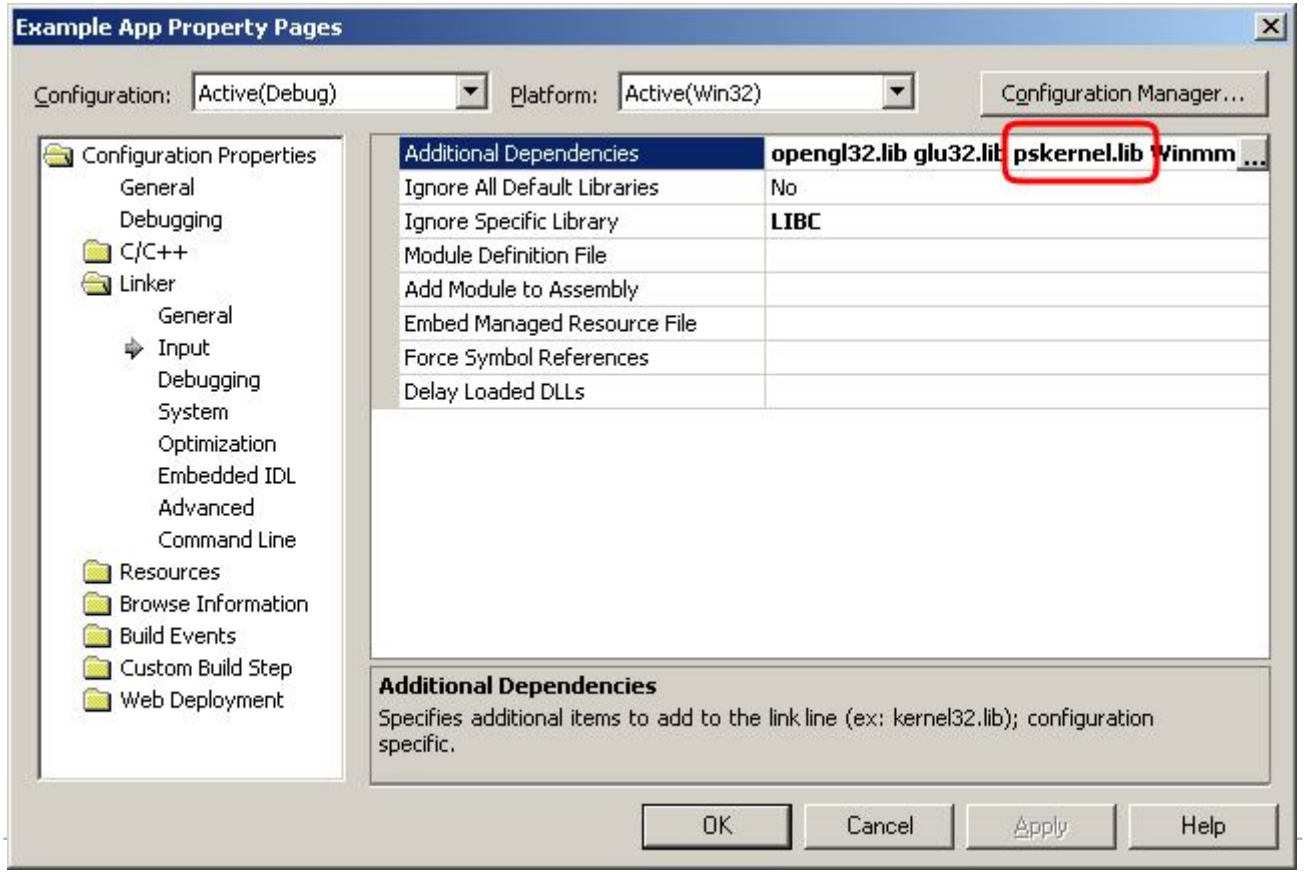
Integrating with MSVC++

- If you are to develop your Parasolid-powered application using Microsoft Visual Studio, you need to integrate Parasolid DLL, LIB, and header files correctly into your project environment.

- This is done as follows:
  - Add the pskernel.lib object library to the MS Visual C++ environment.
  - Add Parasolid header files to the MS Visual C++ environment as an additional include directory.
  - Add #include <parasolid_kernel.h> to any source code files that call PK functionality.
  - Include the file pskernel.dll in either the same folder as your application executable file, or somewhere on your system PATH.

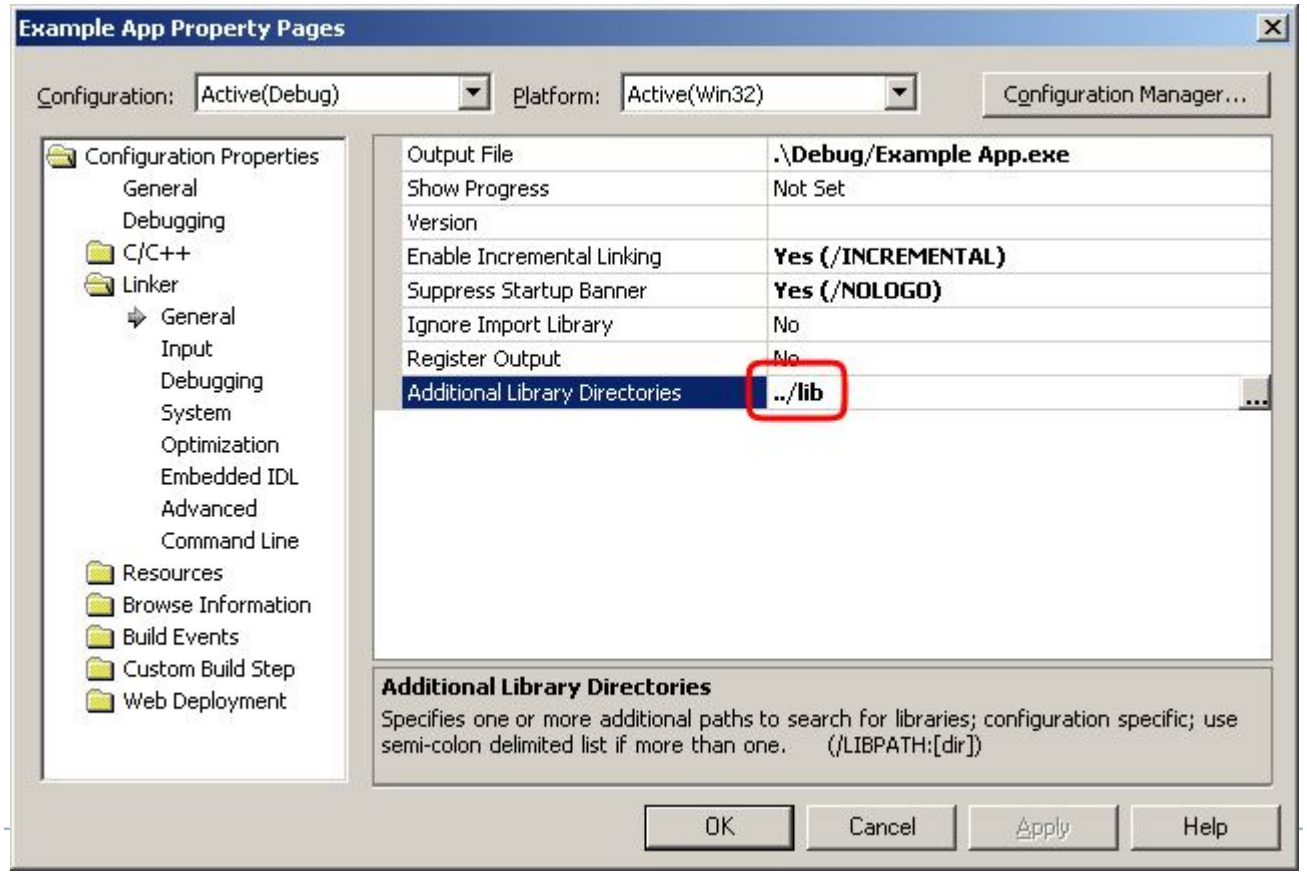
Add pskernel.lib

- Menu > Project > Example App Properties > Linker > Input > Additional Dependencies
  - Add pskernel.lib to the list



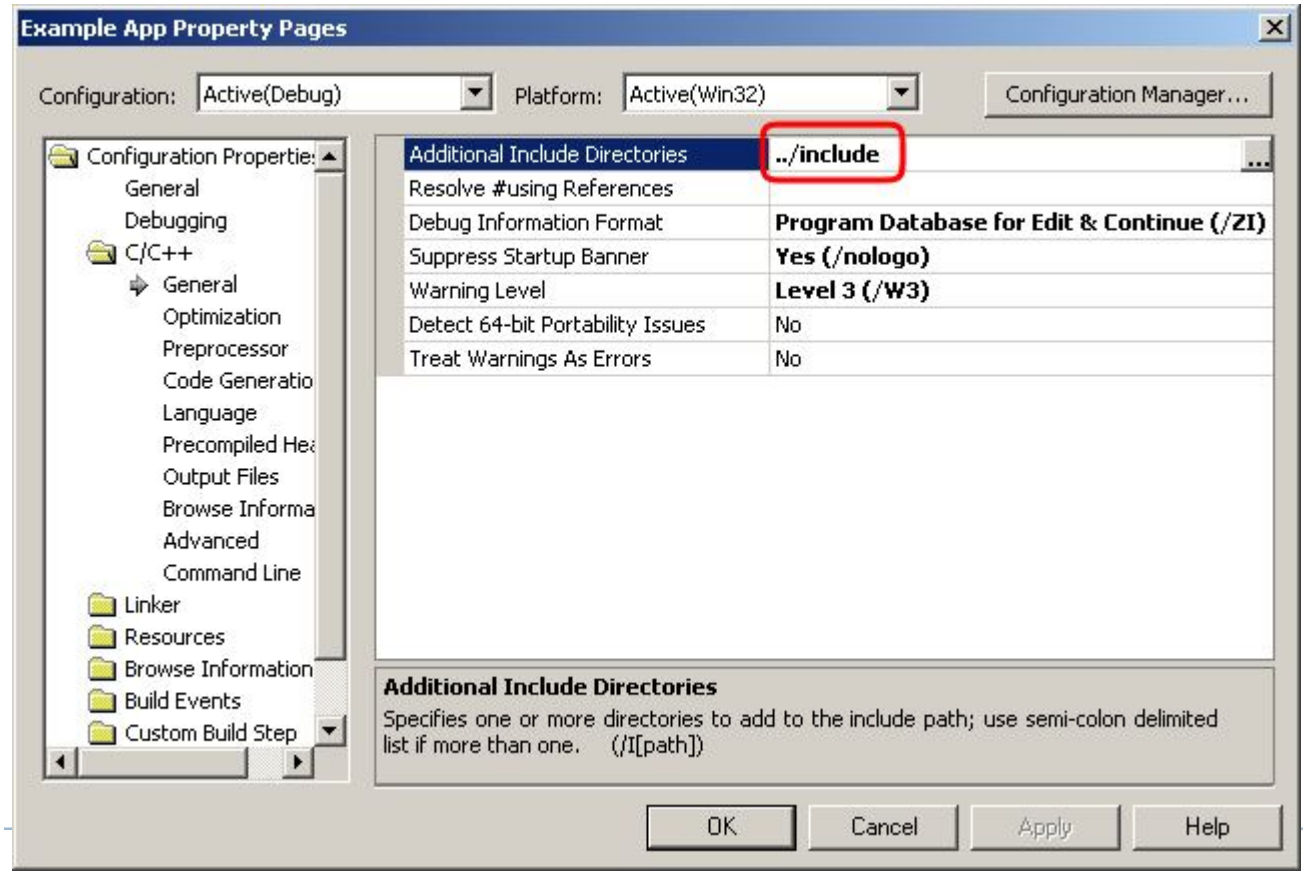
Add pskernel.lib

- Menu > Project > Example App Properties > Linker > General > Additional Library Directories
  - Add the pathname of the directory containing pskernel.lib
  - In this example, you can add ../Lib



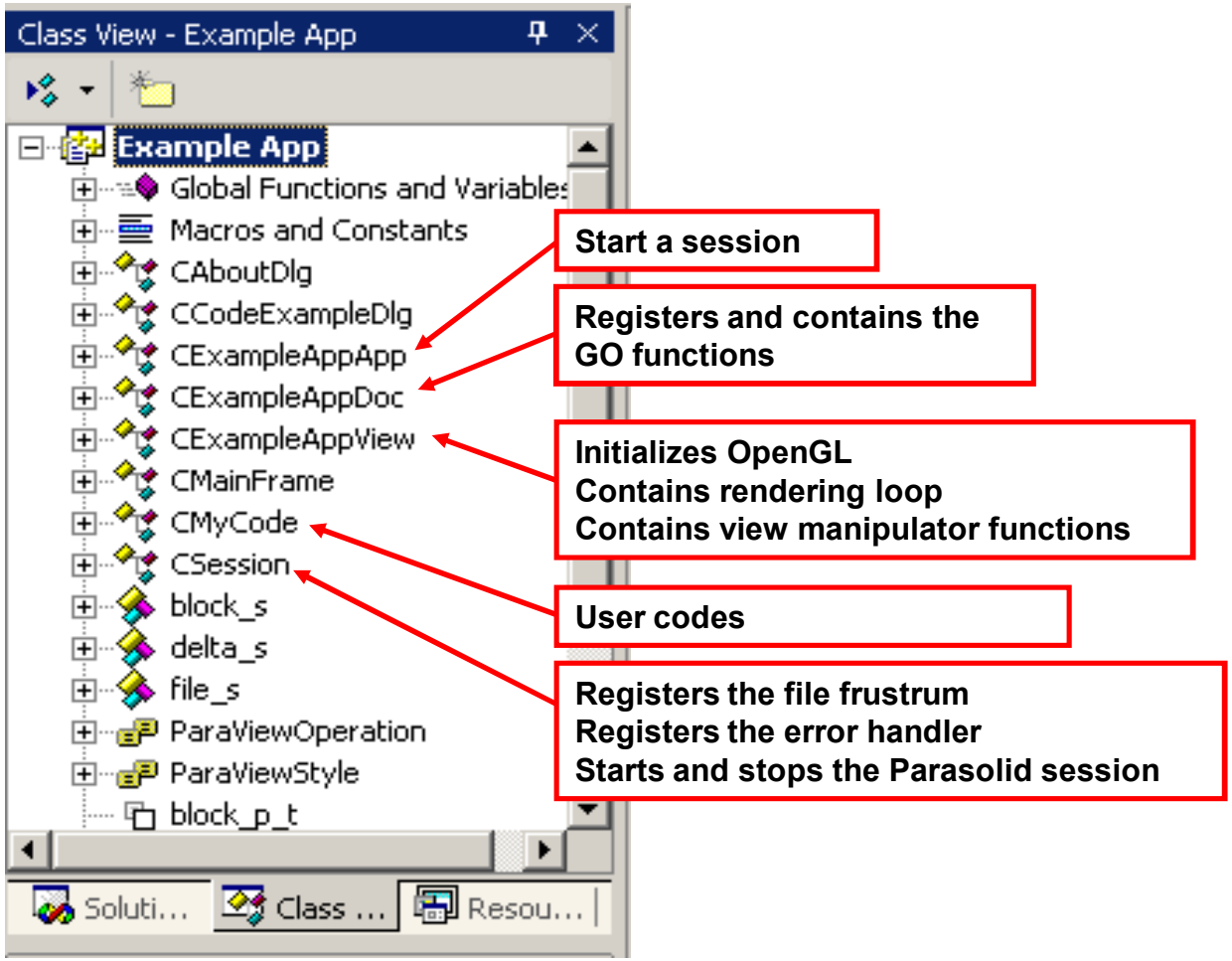
Specify include directory

- Menu > Project > Example App Properties > C/C++ > General > Additional Include Directories
  - Add the pathname of the directory containing parasolid_kernel.h
  - In this example, you can add ../Include



Class Structure

- Class view



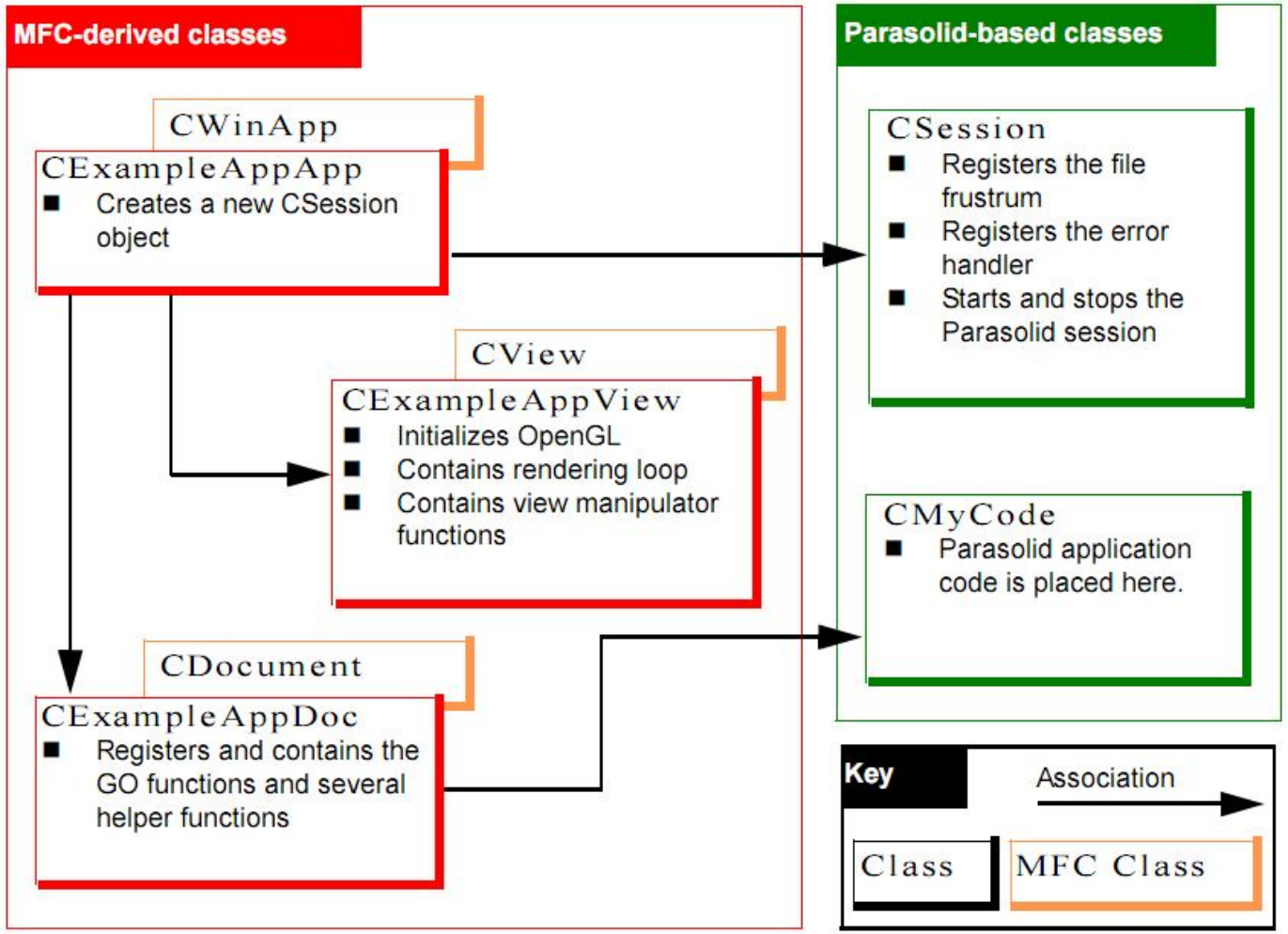
Class Structure

- The Example Application contains the following 7 classes

Classes	Description
CMyCode	This is the class where you should add your own Parasolid application code.
CSession	This class incorporates the code responsible for starting and stopping Parasolid. It registers the frustrum and starts the session as well as registering the error handler. See Chapter 4, "The Example Application", in *Getting Started With Parasolid*, for more details.
CAboutDlg CExampleAppApp CExampleAppDoc CExampleAppView CMainFrame	These classes are part of the MFC framework. The graphical output (GO) functions and helper functions are in CExampleAppDoc, while the rest are in CExampleAppView. See Chapter 4, "The Example Application", in *Getting Started With Parasolid*, for more details. You can ignore CAboutDlg (which contains code for the application's About dialog) and CMainFrame (which contains only MFC-generated code).

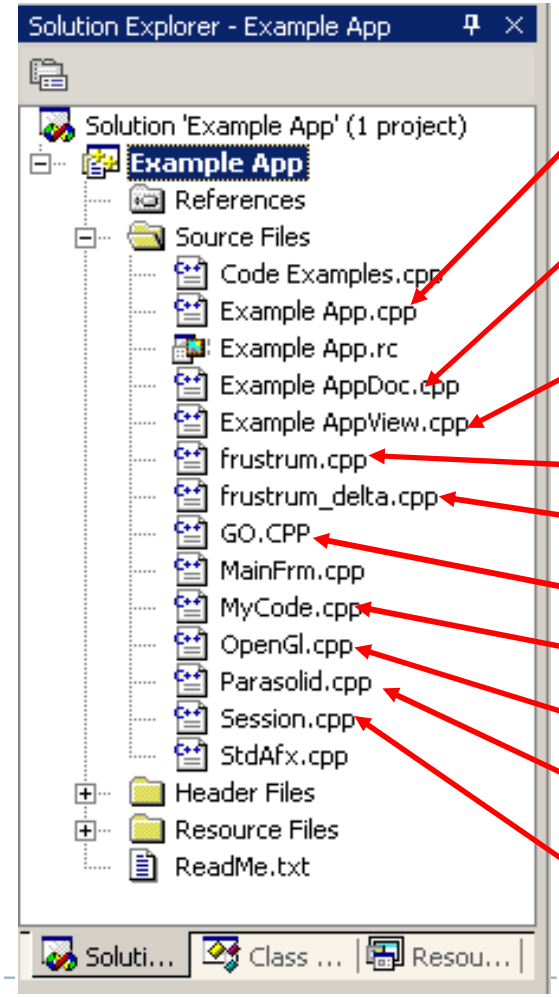
Class Structure

- The relationships between the classes and the functional areas



File Structure

- Solution Explorer



Standard MFC Initialization Creates a new CSession object

Registers graphics frustrum functions
Updates part and geom information

MFC created file
Initializes, Updates and manipulates view
Handles windows messages : mouse events and UI events

File based frustrum functions

Delta functions

GO functions

User codes

OpenGL initialization

Rendering loop : rerender, rotate, zoom, pan

Registers the file frustrum
Registers the error handler
Starts and stops the Parasolid session

File Structure

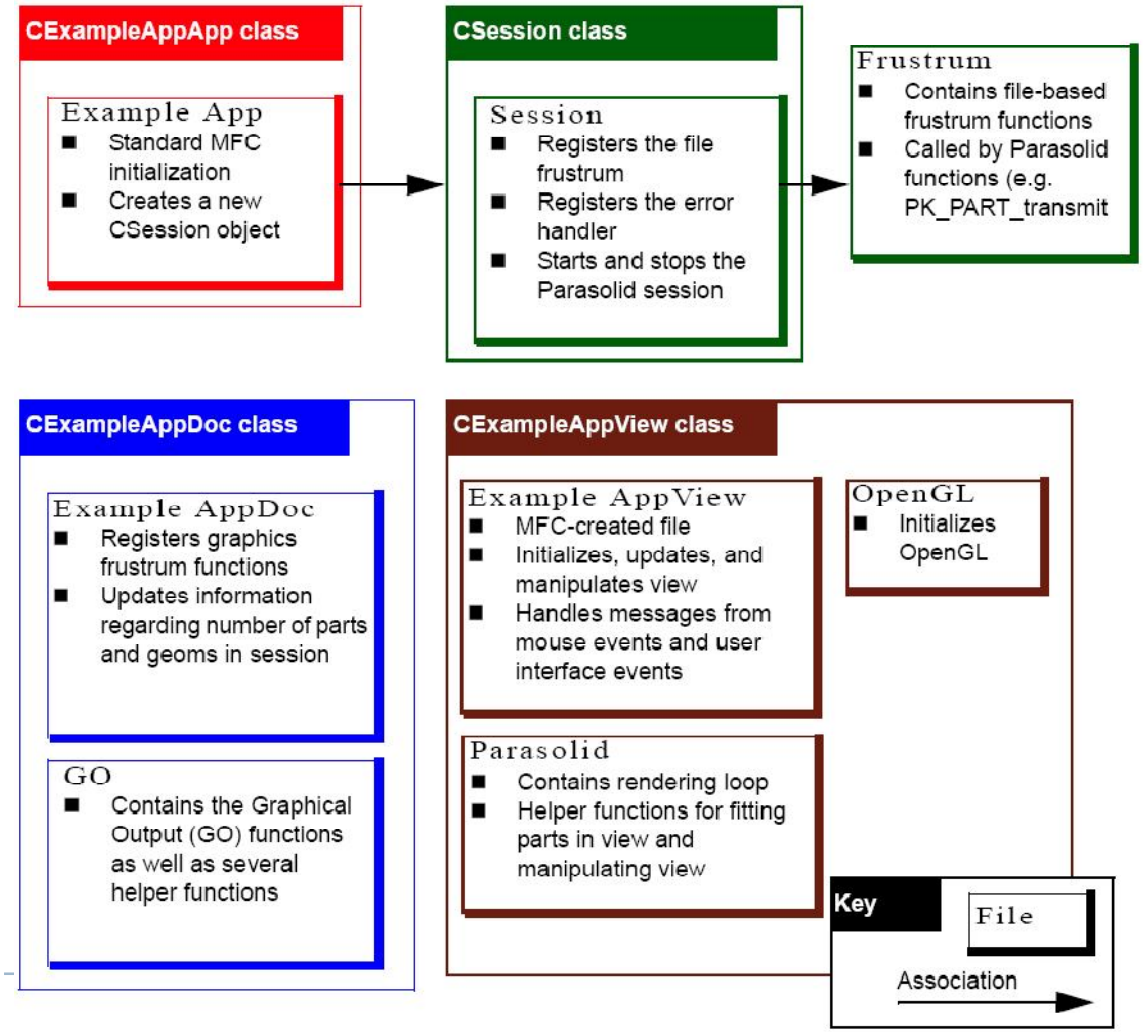
- The classes in the Example Application are split across the following files, containing the functionality

File	Description
Example App	Initializes the MFC application. Creates a new CSession object.
Example AppDoc	Initializes the document and registers the graphical output (GO) frustrum functions. Stores a copy of the parts and geometries in the session.
Example AppView	MFC code for initializing the code and handling window events.
MainFrm	Standard MFC file.
GO	Part of the CExampleAppDoc class. It contains the GO functions as well as various helper functions.

File	Description
OpenGL	Part of the CExampleAppView class. It initializes OpenGL.
Parasolid	Part of the CExampleAppView class. It contains the rendering loop.
Session	Registers the file frustrum and error handler, and starts and stops Parasolid.
MyCode	Contains the class encapsulation for the Parasolid application code you write.
Frustrum	Contains file based frustrum functions.

File Structure

- Overview of the files in the Example Application



1. Add a session object

- CExampleAppApp instantiates a CSession class through the m_session member variable.

```
class CExampleAppApp : public CWinApp
{
        // This creates an instance of m_session which in turn sets and
        // starts up Parasolid. For further details see Csession

        CSession m_session;
```

- CSession contains the functions required for starting up and stopping Parasolid.
  - Creates a new frustrum object and registers the following frustrum functions

2. Register frustrum functions

- These are the minimum frustrum functions that must be registered
- Registers the frustrum that was just created.

```
BOOL CSession::Start()
{
        BOOL ok = TRUE;
        // Register frustrum functions
        // Note: the GO functions are registered in CExampleAppDoc
        PK_SESSION_frustrum_t fru;
        PK_SESSION_frustrum_o_m( fru );

        fru.fstart = StartFrustrum;
        fru.fabort = AbortFrustrum;
        fru.fstop  = StopFrustrum;
        fru.fmallo = GetMemory;
        fru.fmfree = ReturnMemory;
        fru.ffoprd = OpenReadFrustrumFile;
        fru.ffopwr = OpenWriteFrustrumFile;
        fru.ffclos = CloseFrustrumFile;
        fru.ffread = ReadFromFrustrumFile;
        fru.ffwrit = WriteToFrustrumFile;

        VERIFY( PK_SESSION_register_frustrum( &fru ) == PK_ERROR_no_errors);
```

3. Register error handler and Start a session

- Registers the function PKerrorHandler as an error handler.
- Error handlers can be registered and unregistered at any time in the session.
- Starts the session by calling PK_SESSION_start()

```
BOOL CSession::Start()
{
        //… omitted .../!

        // Register Error Handler
        PK_ERROR_frustrum_t errorFru;
        errorFru.handler_fn = PKerrorHandler;
        VERIFY( PK_ERROR_register_callbacks( errorFru ) == PK_ERROR_no_errors );

        // Starts the modeler
        PK_SESSION_start_o_t options;
        PK_SESSION_start_o_m( options ) ;
        PK_SESSION_start( &options );
```

4. Register GO functions

- The GO functions are registered in CExampleAppDoc::OnNewDocument()

```
BOOL CExampleAppDoc::OnNewDocument()
{
        //… omitted …//

        // Register the graphics frustrum functions
        PK_SESSION_ask_frustrum( &m_oldfrustrum );
        m_frustrum = m_oldfrustrum;
        m_frustrum.goopsg = CopenSegment;
        m_frustrum.goclsg = CcloseSegment;
        m_frustrum.gosgmt = CoutputSegment;

        PK_SESSION_register_frustrum( &m_frustrum );

        return TRUE;
}
```

5. Initialize view-specific information

- Initializing view-specific information is handled by the CExampleAppView constructor.

```
CExampleAppView::CExampleAppView()
{
        // Initialize facet options
        PK_TOPOL_facet_mesh_o_m( m_facetOptions.control );
        PK_TOPOL_render_facet_go_o_m( m_facetOptions.go_option );
        m_facetOptions.go_option.go_normals = PK_facet_go_normals_yes_c;

        // Initialize line options
        PK_TOPOL_render_line_o_m( m_lineOptions );
        m_lineOptions.planar = PK_render_planar_attrib_c;
        m_lineOptions.radial = PK_render_radial_attrib_c;
        m_lineOptions.param  = PK_render_param_attrib_c;

        // Initialize geometry line options
        PK_GEOM_render_line_o_m( m_geomlineOptions );
        m_geomlineOptions.is_curve_chord_tol = PK_LOGICAL_true;
```

6. Initialize OpenGL

- OpenGL is initialized by CExampleAppView::OnInitialUpdate()

```
void CExampleAppView::OnInitialUpdate()
{
	m_updateNeeded = TRUE;

	if (m_pDC == NULL)
		InitOpenGL();

	MakeCurrent( TRUE );

	CExampleAppDoc* doc = GetDocument();
	doc->m_view = this;  // Store current view in document class

	SetLighting( doc->m_viewStyle );

	Invalidate();

}
```

7. Executing my code

- The function OnTestBtn calls CMyCode::RunMyCode to execute any Parasolid application code you have placed there.

```
void CExampleAppDoc::OnTestBtn()
{
        //… omitted …//
        if (m_ngeoms > 0)
                {
                        PK_ENTITY_delete( m_ngeoms, m_geoms );
                        PK_MEMORY_free( m_geoms );
                }
                PK_MEMORY_free( partitions );
        }

        CMyCode test;
        int result = test.RunMyCode( ++step );

         if (result > 0)
        {
                if (step != 1) AfxMessageBox( "Finished. Press \"!\" again to rerun." );
                step = 0;
        }
        //… omitted …//
```

8. Store geometry and Update view

- The parts and geometries in the session are extracted and stored in the document.

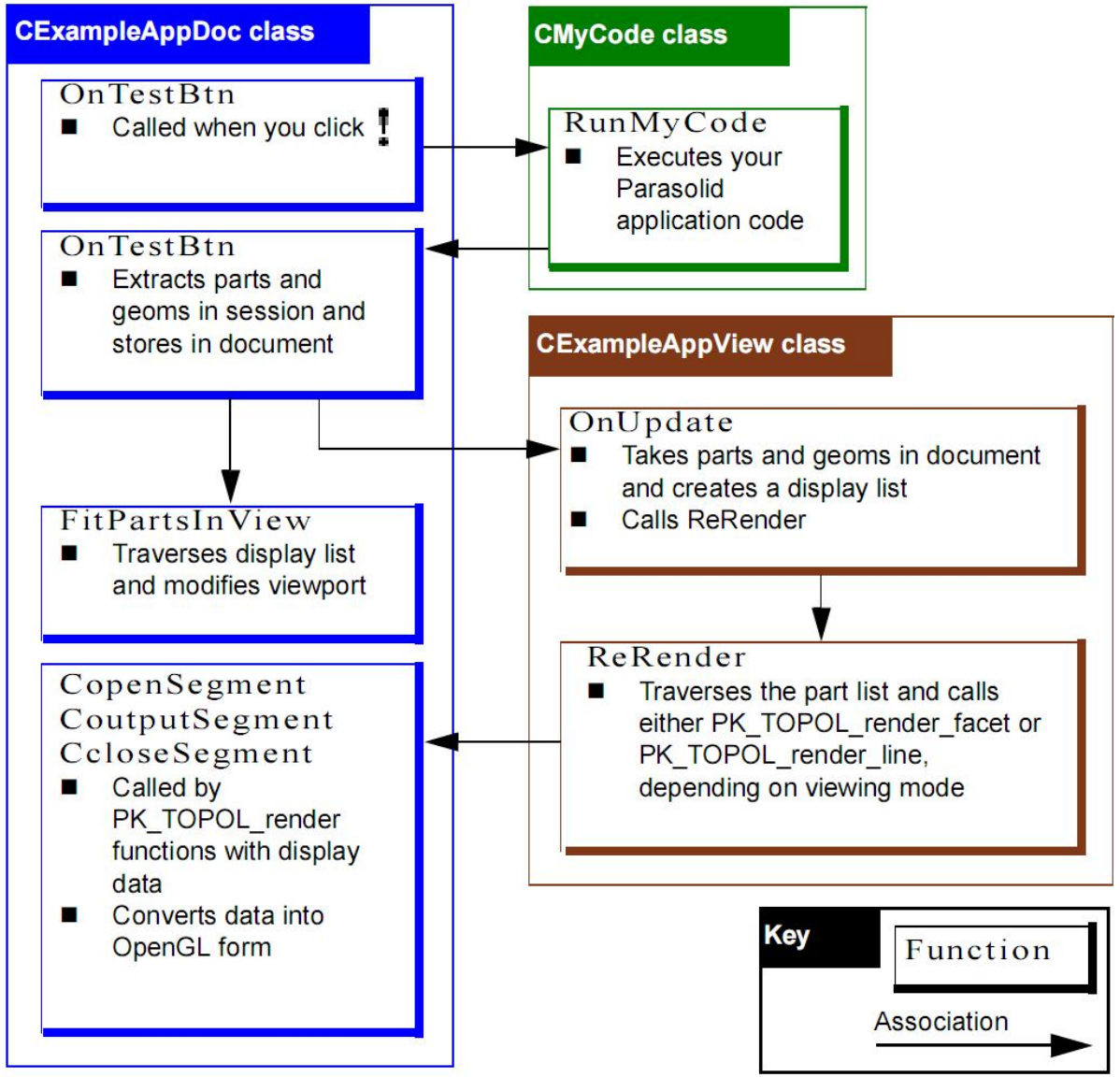
```
void CExampleAppDoc::OnTestBtn()
{
        //… omitted …//
        else if ( result == -1 )
        {
                step = 0;
                return;
        }

        int nPartitions;
        PK_PARTITION_t *partitions = NULL;
        PK_SESSION_ask_parts( &m_nparts, &m_parts );

        // Get all the geometry in the partition.
        PK_SESSION_ask_partitions( &nPartitions, &partitions );
        ASSERT( nPartitions == 1 );
        PK_PARTITION_ask_geoms( partitions[0], &m_ngeoms, &m_geoms );

        UpdateAllViews( NULL );
        m_view->FitPartsInView();
}
```

9. Display partlist using OpenGL

- CExampleAppView::OnDraw clears the screen and traverses the OpenGL display list and displays it to screen.

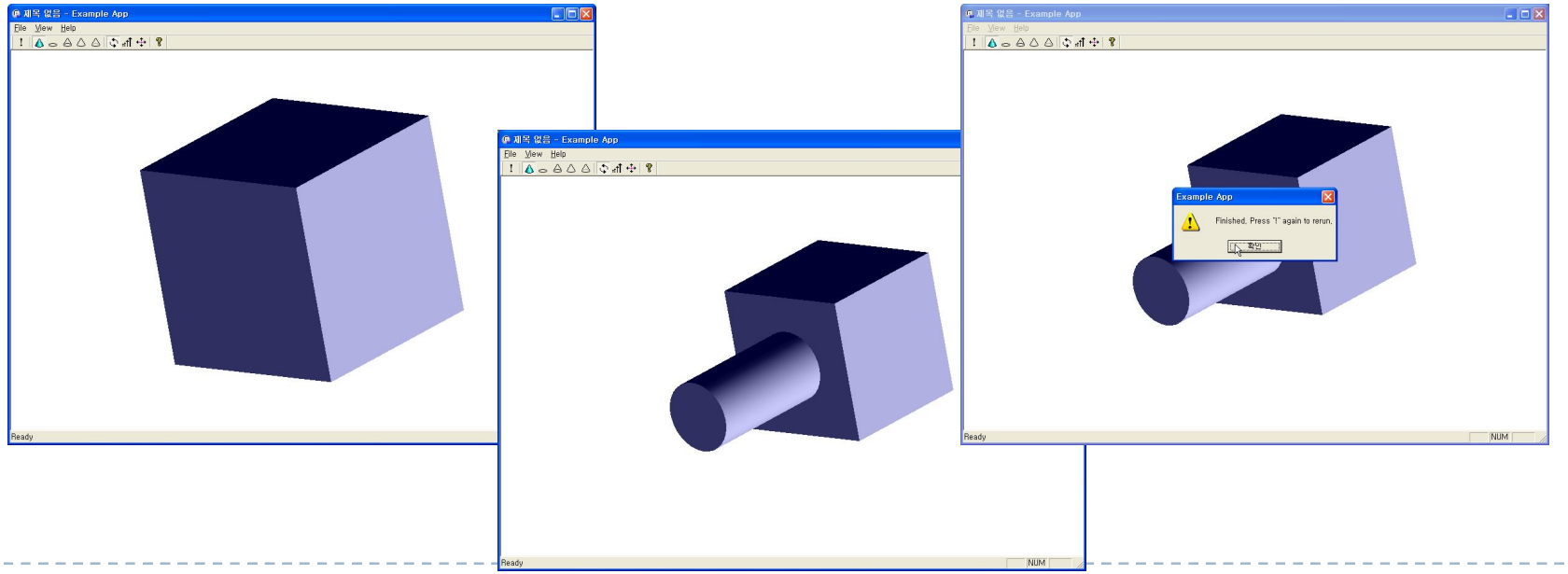
```
void CExampleAppView::OnDraw(CDC* pDC)
{
	//… omitted …//
	VERIFY_GL( glTranslated( -pDoc->m_viewCentre.coord[ 0 ],
	                         -pDoc->m_viewCentre.coord[ 1 ],
	                          -pDoc->m_viewCentre.coord[ 2 ] ) );

	if ( m_geomList.IsEmpty() == FALSE || m_partList.IsEmpty() == FALSE )
	{
		VERIFY_GL( glCallList( m_partDisplaylist ) );
	}

	VERIFY_GL( glFlush() );
	VERIFY_GL( glPopMatrix() );

	SwapBuffers( pDC->m_hDC );

	pDC->RealizePalette();
	MakeCurrent( FALSE );
}
```

10. Order of events after clicking



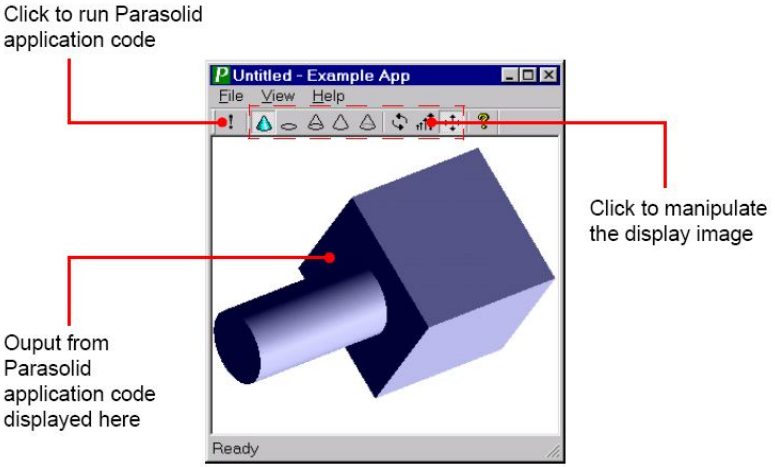
Compile & Run

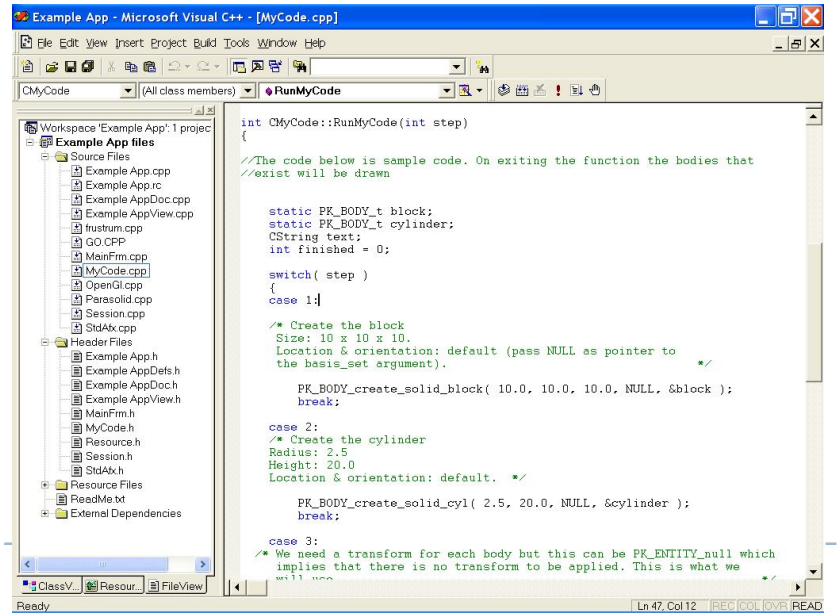
- Build the Example Application
  - Menu > Build > Rebuild Solution
- Execute the Example Application
  - Need the Parasolid Library file(PSKERNEL.dll)
  - In this case, you can copy pskernel.dll into debug folder



Easy way for developing applications

- The simplest way to create your Own Parasolid project
  - Copy the Parasolid Example Project to your directory and change it according to your requirements
  - You can utilize its project setting
  - Add your codes in CMyCode::RunMyCode()





To be continued..

The END