Parasolid V13.0

Using the Example Application

June 2001

Important Note

This Software and Related Documentation are proprietary to Unigraphics Solutions Inc.

Unigraphics Solutions™

Parker's House
46 Regent Street
Cambridge CB2 1DP
UK
Tel: +44 (0)1223 371555
Fax: +44 (0)1223 316931
email: ps-support@ugs.com
Web: www.parasolid.com

Trademarks

Parasolid is a trademark of Unigraphics Solutions Inc.

HP and HP-UX are registered trademarks of Hewlett-Packard Co.

SPARCstation and Solaris are trademarks of Sun Microsystems, Inc.

Alpha AXP and VMS are trademarks of Digital Equipment Corp.

IBM, RISC System/6000 and AIX are trademarks of International Business Machines Corp.

OSF is a registered trademark of Open Software Foundation, Inc.

UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

Microsoft Visual C/C++ and Window NT are registered trademarks of Microsoft Corp.

Intel is a registered trademark of Intel Corp.

Silicon Graphics is a registered trademark, and IRIX a trademark, of Silicon Graphics, Inc.

All other trademarks are the property of their respective owners.

Table of Contents

1 Introduction 5

1.1 Installing the Example Application **5**
1.2 Building the Example Application **5**

2 Running the Example Application 7

2.1 Buttons in the toolbar **8**

3 Example Application Source Code 9

3.1 Class structure **9**
3.2 File structure **10**
3.3 Initializing Parasolid **13**
3.4 Executing calls to Parasolid **14**

Introduction 1

The example application demonstrates a very simple Parasolid-powered application within a Windows framework. It is built using the Microsoft Visual C/C++ development environment within an MFC framework, and uses OpenGL for display to the screen. The application is intended as a framework within which to demonstrate various aspects of Parasolid, and does not form a complete example of how to write a Parasolid-powered application.

Within the framework of the application, you can add and execute Parasolid application code (i.e. code that includes calls to Parasolid functionality), and display the results on screen. The code you add can either be executed all at once or in stages, using a `case` statement. At the end of each stage the contents of the Parasolid session are displayed to screen.

1.1 Installing the Example Application

The files required for the Example Application can be found on the Parasolid CD for PC Platforms, in the `EXAMPLE_APPLICATION` folder. You will need a ZIP utility such as WinZip in order to extract the files from the archive.

To install the Example Application on your computer:

- Open the file `\EXAMPLE_APPLICATION\EXAMPLEAPP.ZIP` on the CD.
- Extract all the files in this archive to a folder on your computer, making sure that you preserve the folder names specified in the archive.

See Section 1.2 for instructions on building the Example Application using the extracted files.

1.2 Building the Example Application

You need Microsoft Visual C++ Version 6.0 with Service Pack 4 to successfully compile and run this application. The example application has been built on the latest FCS version of Parasolid and requires that version or later, and uses the OpenGL graphics library for displaying part models.

For the executable to run successfully please ensure that your display color setting is set to “true color”.

To load the project into MSVC++, open the file `Source\Example App.dsw`.

To build the Example Application for the first time, choose **Build > Rebuild All**. Rebuilding the whole project removes any anomalous compilation errors that may be present on first extracting the files from the archive.

Note: On some installations there may be a problem with the VERIFY_GL macro asserting. If this occurs then you should obtain and install the latest graphics cards drivers from the manufacturer. Alternatively specify the preprocessor definition NOCHECK_GL (choose **Project > Settings**, click the C/C++ tab, and edit the Preprocessor definitions text box) and recompile.

Running the Example Application

2

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

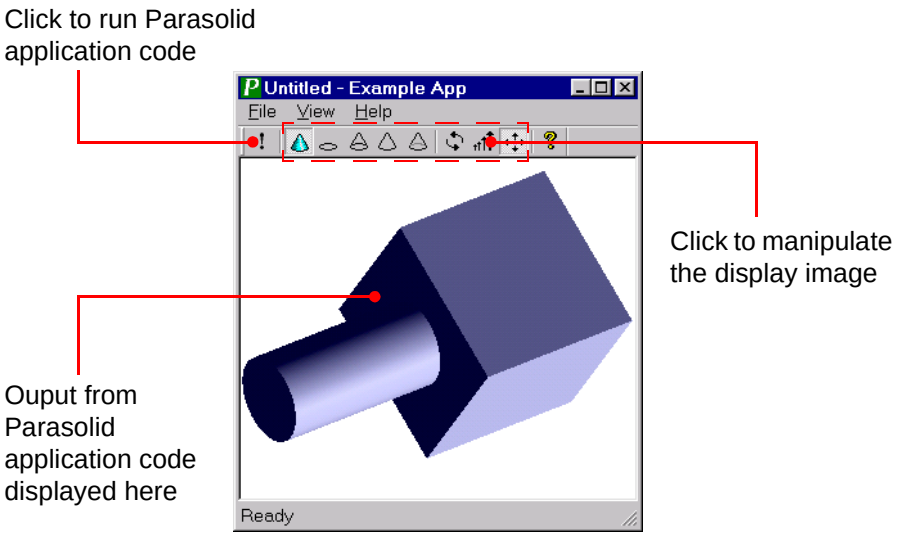

Figure 2–1 The Parasolid Example Application

2.1 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.

Example Application Source Code

3

3.1 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).

The relationships between the classes and the functional areas of each are detailed in Figure 3–1.

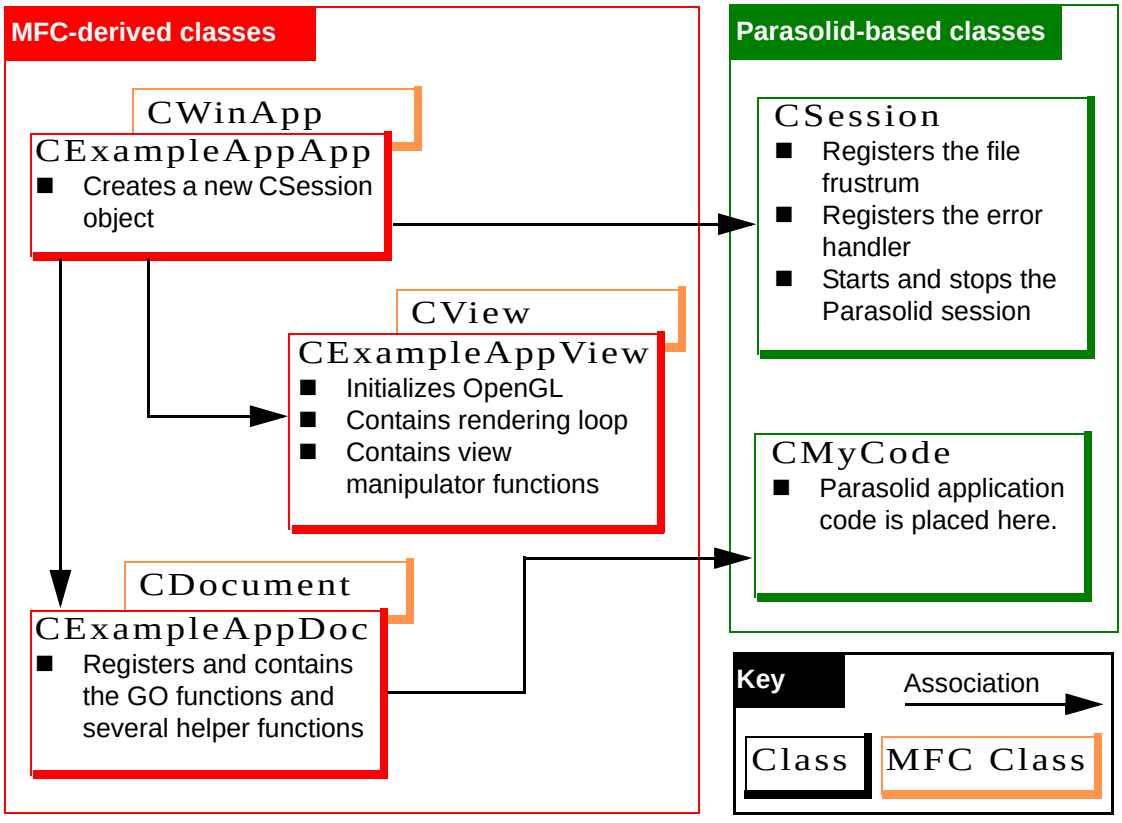

Figure 3–1 Relationships between the classes in the Example Application

3.2 File structure

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

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.

File	Description
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.
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.

The relationships between these files and the functionality they contain are shown in Figure 3–2.

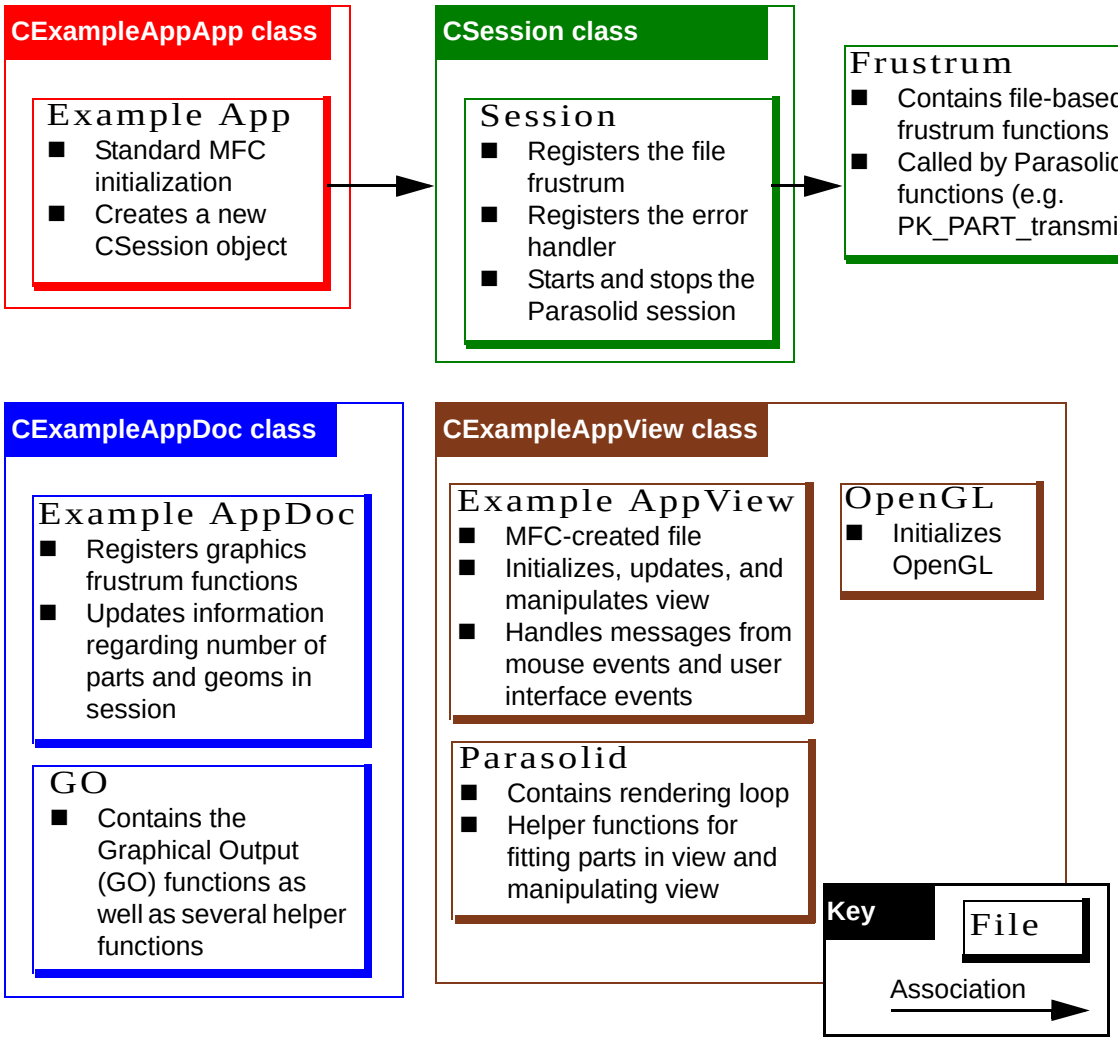

Figure 3–2 Overview of the files in the Example Application

3.3 Initializing Parasolid

The Example Application initializes Parasolid as follows:

- CExampleAppApp instantiates a CSession class through the `m_session` member variable.
- CSession contains the functions required for starting up and stopping Parasolid. It does the following:
  - Creates a new frustrum object and registers the following frustrum functions:

    StartFrustrum (`fstart`)
    AbortFrustrum (`fabort`)
    StopFrustrum (`fstop`)
    GetMemory (`fmallo`)
    ReturnMemory (`fmfree`)
    OpenReadFrustrumFile (`ffoprd`)
    OpenWriteFrustrumFile (`ffopwr`)
    CloseFrustrumFile (`ffclos`)
    ReadFromFrustrumFile (`ffread`)
    WriteToFrustrumFile (`ffwrit`)

    These are the minimum frustrum functions that must be registered before the session is started. See Chapter 3, “Supplying A Frustrum”, in *Getting Started With Parasolid*, for an introduction to the frustrum functions that you need to supply, and what they must do.

    The graphical output functions are not registered at this stage, because they require access to document and view classes that have not been created yet. They are registered in CExampleAppDoc instead.

  - Registers the frustrum that was just created.
  - Registers the function PKerrorHandler as an error handler. This function is called and passed details just before a failing PK function is about to return. For simplicity, the error handler in the Example Application just

echoes any error messages to the screen. Error handlers can be registered and unregistered at any time in the session.
    - Starts the session by calling PK_SESSION_start.
- A new document is created and initialized as follows:
    - The document is created and MFC-specific initialization is handled by the MFC framework.
    - Specific initialization (such as setting the default view) is handled by the CExampleAppDoc constructor
    - The GO functions are registered in CExampleAppDoc::OnNewDocument, which overrides CDocument::OnNewDocument.
- A new view is created as follows:
    - The view is created and attached to the document using the MFC framework.
    - Initializing view-specific information is handled by the CExampleAppView constructor.
    - OpenGL is initialized by CExampleAppView::OnInitialUpdate.

3.4 Executing calls to Parasolid

The following events occur when you click ! in the Example Application:

- The function OnTestBtn calls CMyCode::RunMyCode to execute any Parasolid application code you have placed there.
- The parts and geometries in the session are extracted and stored in the document.
- OnTestBtn calls CExampleAppView::OnUpdate() (via UpdateAllViews) to create a display list (a list of entities to display)
- OnUpdate calls CExampleAppView::ReRender() which, together with calls to the GO functions, renders the display.
- Finally, TestBtn calls CExampleAppView::FitPartsInView() to fit all the entities within the current window.

ReRender controls the rendering loop in the Example Application.

- When rendering in shaded view, it calls PK_TOPOL_render_facet and passes it all the entities in the parts list.
- When rendering in any line views, it calls PK_TOPOL_render_line and passes it all the entities in the parts list.
- It calls PK_GEOM_render_line and passes it all the entities in the geometry list.

Each Parasolid rendering function then calls GO functions whch convert the rendering data into OpenGL form and populate the OpenGL display lists. Finally, CExampleAppView::OnDraw clears the screen and traverses the OpenGL display list and displays it to screen.

This sequence of events is illustrated in Figure 3–3.

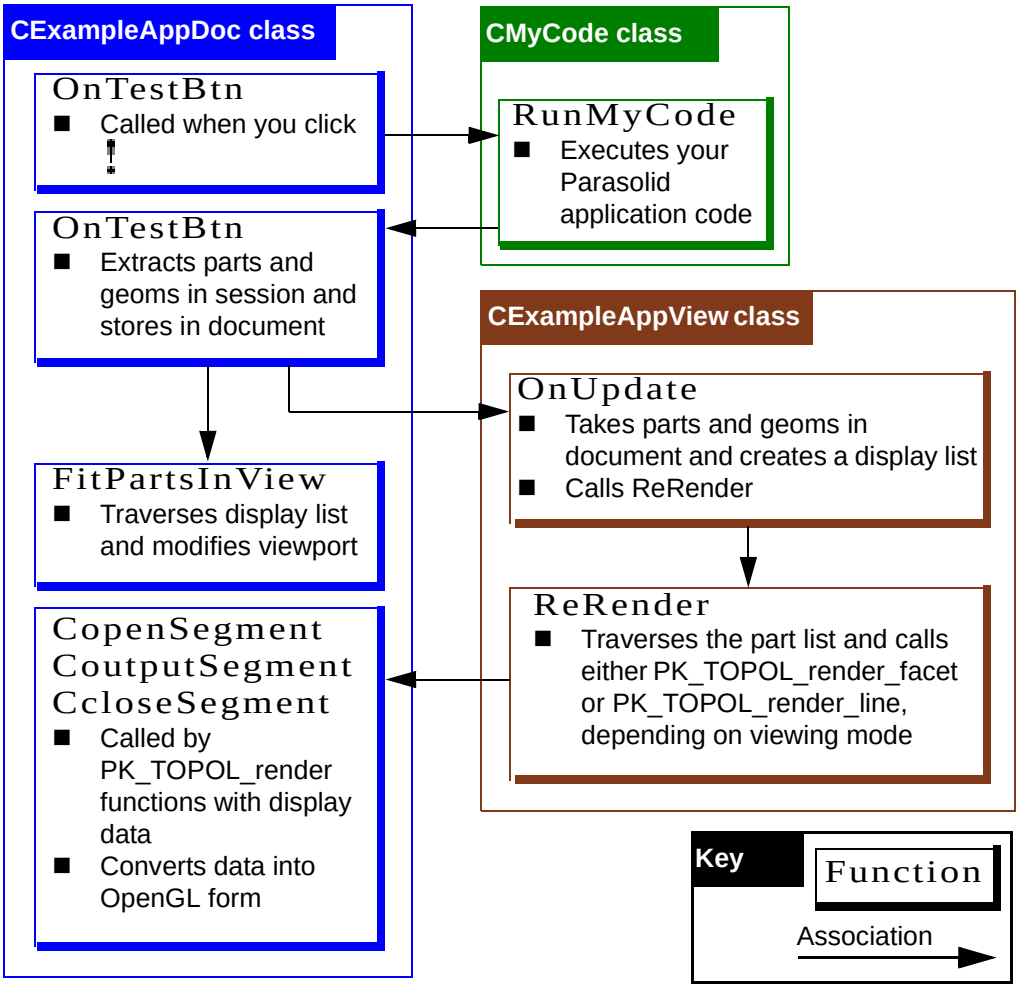

Figure 3–3 Order of events after clicking !