Parasolid V13.0

Release Notes

June 2001

Important Note

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

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.

Table of Contents

A

Introduction 1

1.1 Introduction

Parasolid is a copyright product of Unigraphics Solutions Inc. This document covers details of the release of Parasolid and supporting utilities, and retrieval of the product onto the following platforms:

- Intel-based PCs running Windows NT, Windows 2000, or Linux
- AMD Athlon-based PCs running Windows NT, Windows 2000, or Linux
- HP 9000 workstations running the HP-UX operating system
- SPARCstations 5, 10, 20 and Ultra, running the Solaris operating system
- Compaq Alpha workstations running the Tru64 operating system
- IBM RISC System/6000 running the AIX operating system
- Silicon Graphics, running the IRIX operating system

It assumes a basic knowledge of the operating system for the relevant platform (creating and moving directories, deleting and listing files for example).

1.2 This Document

The chapters of this document describe:

- what this release comprises (“The Product”)
- how to retrieve the product from CD (“Installing Parasolid”)
- what files are included in the release (“Parasolid Release Area”)
- how to run the product (“Using Parasolid”)
- what enhancements (and known problems) are introduced in this version (“Functional Enhancements”)
- how the product interface has changed in this version (“Interface Changes”)
- in what operating system environment the product was created (“Environment”)

1.3 Platform-specific information

Throughout this manual, text and instructions that are specific to a particular platform or group of platforms are referred to as follows:

Label	Platform
HPPA HPUX	For HP 9000 workstations running the HP-UX operating system
SPARC Solaris	For SPARCstations (5, 10, 20, and Ultra) running the Solaris operating system
AXP OSF	For Compaq Alpha workstations running the Tru64 operating system
RS6000 AIX	For the IBM RISC System 6000 running the AIX operating system
R4000 IRIX	For Silicon Graphics platforms running the IRIX operating system
Intel NT	For Intel-based or Athlon-based PCs running Windows NT or Windows 2000
Linux	For Intel-based or Athlon-based PCs running Linux
UNIX	Includes all, or the majority of, the following platforms: ■ Intel-based or Athlon-based PCs running Linux ■ HP 9000 platform running the HP-UX operating system ■ SPARCstation platforms running the Solaris operating system ■ Compaq Alpha workstations running the Tru64 operating system ■ Silicon Graphics platform running the IRIX operating system ■ IBM RISC System 6000 platform running the AIX operating system
NT	Includes the following platforms: ■ Intel-based PCs running Windows NT or Windows 2000 ■ AMD Athlon-based PCs running Windows NT or Windows 2000 ■ Compaq Alpha PCs running Windows NT

The Product 2

2.1 Release Contents

This release of Parasolid consists of the following CDs:

PC Platforms	The Parasolid library, driver program, acceptance tests, and other files necessary for the programs to run, for the Intel NT and Linux platforms. This CD also contains the PS/Workshop application for viewing and manipulating part files, and the Example Application that is discussed in *Getting Started With Parasolid*.
UNIX Software	The Parasolid library, driver program, acceptance tests, and other files necessary for the programs to run, for all supported UNIX platforms
Documentation	The Parasolid On-line Documentation Web in HTML format
Parasolid Interoperability Tools	Evaluation versions of PS/Translators and PS/Bodyshop.
ParaHOOPS 3D Part Viewer	A third-party part viewer for Parasolid parts.

2.2 Identification

The full version number for this release is 13.00.151.

The schema name for this release is SCH_13006.

2.3 Compatibility

Parts created in earlier versions of Parasolid are upward compatible with V13.0.

2.3.1 Filename extensions on NT

Up to V6.1, KID and the dummy Frustrum always created textual part files with the extension ".X_T" and accessed textual schema files with the extension ".S_T" (with a similar naming convention for other file guises and binary format files).

Since V6.2, KID and the dummy Frustrum test whether a file resides on a DOS style FAT device or a long name NTFS type device before opening the file, and act accordingly:

- Files on NTFS devices use the same 7 character extensions (“.xmt_txt”, “.sch_txt” etc.) as on the other platforms.
- Files on DOS style FAT devices use the same 3 character extension as at V6.1. They are shown in the following table in upper case for clarity, though the case is ignored.

This table shows the relationships between the FAT and NTFS names (files can simply be renamed when transferring between the different systems):

	FAT	NTFS
part	.X_T	.xmt_txt
schema	.S_T	.sch_txt
journal	.J_T	.jnl_txt
snapshot	.N_T	.snp_txt
partition	.P_T	.xmp_txt
delta	.D_T	.xmd_txt
binary	.*_B	.***_bin

Installing Parasolid 3

3.1 Disk Space Requirements

3.1.1 Software

To install this release of Parasolid you will require the following disk space:

Intel NT	58.5 Mbytes
Intel NT (Pentium 4)	59 MBytes
Linux	49.5 Mbytes
HPPA HPUX (32-bit)	107.5 Mbytes
HPPA HPUX (64-bit)	120 Mbytes
SPARC Solaris (32-bit)	94 Mbytes
SPARC Solaris (64-bit)	96.5 Mbytes
AXP OSF	94 Mbytes
RS6000 AIX	87 Mbytes
R4000 IRIX	98 Mbytes

3.1.2 Documentation

Full installation, with search and PDF files	111 Mbytes
Search files	62.5 Mbytes
PDF files	36 Mbytes

If you are installing the documentation to a local hard disk, you can save on disk space by omitting the search files or PDF files. See Section 3.3, “Documentation”.

3.2 Copying files to a local disk

Follow these instructions to copy the Parasolid distribution onto a local hard disk.

3.2.1 Using the installer (Intel NT only)

If you are running Windows 2000 or Windows NT on an Intel-based platform, you can install Parasolid onto an NT filesystem as follows:

- Insert the CD in the drive. If your computer is set up to run data CDs automatically, the Parasolid installer starts after a few moments.

  If the installer does not start, run the file *X*`:\intel_nt\setup.exe`, where *X* is the drive your CD is mounted on.

- Follow the instructions on screen to install Parasolid Designer and/or Parasolid Communicator onto the filesystem.

Note: By default, Parasolid is installed in `C:\Program Files\Parasolid\kernel\v12.1`, but you can choose a different location from within the installer.

3.2.2 Copying files manually

If you are running on UNIX or Linux, or if you do not want to use the supplied installer, you can copy files to a filesystem as follows:

UNIX or Linux

To copy files to a UNIX or Linux filesystem:

- Insert the CD in the drive.
- Login as superuser.
- Create a mount point for the CD:

  ```
  $ mkdir /cdrom
  ```

- Mount the CD onto the system at the mount point, using the following example command:

Platform	Directory	Example Command
HP 9000 Series 700	`hppa_hpux`	`$ pfs_mount -x lower_case /dev/dsk/clt2d0 /cdrom`
SGI	`r4000_irix`	`$ mount -t /dev/scsi/sc0d710 /cdrom`
Sun	`sparc10_solaris`	(automatic mounting)
IBM	`rs6000_aix`	`$ mount -r -v cdrfs /dev/cd0 /cdrom`
Compaq Alpha PC	`axp_osf`	`$ mount -t cdfs -o noversion /dev/rz3c /cdrom`
Linux	`intel_linux`	`$ mount /dev/cdrom /cdrom -t iso9660`

Warning: On HP: make sure you have run the following daemons prior to this command:
`/usr/sbin/pfs_mountd`
`/usr/sbin/pfsd 4`

- Copy the files from the appropriate directory, as given in the table above. The files are created under your current directory:
  - `cp -r /cdrom/`*`directory`*`/base` to copy Parasolid Designer
  - `cp -r /cdrom/`*`directory`*`/base_alternative` to copy Parasolid Communicator

3.3 Documentation

The Documentation CD contains the Parasolid On-line Documentation in the directory tree "`html/online_docs`". This consists of the following Parasolid manuals, provided in both HTML and PDF format:

- Functional Description
- Getting Started With Parasolid
- PK Interface Programming Reference
- KI Programming Reference
- Downward Interfaces
- Kernel Interface Driver (KID)
- Foreign Geometry User Guide
- Release Notes (this document)

In addition, the Transmit File (XT) Format manual is provided in PDF format only.

You need to have access to a web browser such as Netscape, Internet Explorer, or Opera in order to view the HTML documentation.

3.3.1 Viewing the documentation

You can read the documentation directly from the CD as follows:

NT
- Place the Documentation CD in your CD drive.
- If your computer plays data CDs automatically, the main page of the documentation will be displayed in your internet browser after a short pause.

If your computer does not play data CDs automatically, use Windows Explorer to open the file *X*`:\html\online_docs\index.html`, where *X* is the drive your CD is mounted on.

UNIX or Linux

- Create a mount point for the CD: `$ mkdir /cdrom`
- Mount the CD onto the system at the mount point, using the example commands given in Section , “UNIX or Linux” above.
- Use your web browser to open the file `/cdrom/html/online_docs/index.html`.

3.3.2 Using the PDF documentation

You need a copy of the free Acrobat Reader program from Adobe Systems Inc. to view the PDF documentation. A version that runs on Windows 95, Windows 98, Windows NT, or Windows 2000 is supplied on the Documentation CD. To install this on your computer:

- Place the Documentation CD in your CD-ROM drive (hold down the Shift key if you want to prevent the documentation from displaying while you do this).
- Run the file *X*`:\acroread\ar405eng.exe`, where *X* is the drive your CD is mounted on.

If you use a UNIX-based operating system, or if you want to install a non-English version of Acrobat Reader, you can download a version from `http://www.adobe.com/products/acrobat/readstep.html`.

3.3.3 Copying the documentation to a local hard disk or intranet

If you wish, you can copy the entire documentation set onto a local hard disk, or a disk on your local intranet. To do this, just copy the entire `online_docs` folder to a location of your choice. If you are using UNIX, you will need to mount the CD drive first, as described in Section , “UNIX or Linux” above.

Once you have copied the documentation, create a bookmark or link to the `index.html` file in the `online_docs` directory.

The HTML documentation uses Cascading Style Sheets to control the appearance of the documentation: if your browser does not support CSS, or you do not wish to use it, then you can rename or delete the file `online_docs\ps_doc.css`.

3.3.4 Minimizing disk requirements for documentation

If you do not want to use the search functionality provided in the full documentation set, or if your browser does not support JavaScript, you can reduce the disk space used by the documentation:

- Create a link or bookmark to the file `index2.html` instead of `index.html`
- Remove the `searchfiles` sub-directory from your installation

This will save you approximately 58MB of disk space, but you will not be able to search through the entire documentation set.

You can save more disk space by removing the PDF documentation.

- If you wish to keep a copy of the *XT Format Manual*, copy the file `online_docs\pdf\xt_format.pdf` to a safe place.
- Remove the `pdf` sub-directory from your installation
- If you wish, remove the `pdf_index.html` page from your installation

3.4 Additional CDs

The Parasolid Interoperability Tools and ParaHOOPS 3D Part Viewer CDs contain samples of software for use with Parasolid. Some of this software is provided for evaluation purposes only. Further information on the contents of these CDs can be found in the Read Me files on each CD.

Parasolid Release Area 4

4.1 Contents of the Release Area

As described in Chapter 3, “Installing Parasolid”, the top level of each Parasolid release CD contains a directory for each platform available on that CD. For example, the PC Platforms CD contains two directories: `intel_nt`, and `intel_linux`.

Inside each directory is a `base` directory that contains all the files necessary to install and use Parasolid for that platform.

On NT platforms, there is an additional `base_alternative` directory that contains all the files necessary to install and use Parasolid Communicator.

On the SPARC Solaris platform, there is an additional `base_sun64` directory that contains all the files necessary to install and use Parasolid on 64-bit Solaris machines. The contents of the `base_sun64` directory are identical to the contents of the `base` directory for SPARC Solaris.

This chapter describes the contents of these directories in some more detail. Apart from some platform-dependent filenames (noted below), the contents are essentially the same for each platform.

Note: This chapter describes the contents as they are structured on the release CD. Your own installation of Parasolid will not necessarily reflect this structure.

4.1.1 Privileges

You must make sure that all Parasolid users are able to read these files once they have been copied from the CD. In addition, users will need to be able to write files to the schema directory.

4.1.2 The ‘base’ directory

The `base` directory contains all the files needed to install and use the full Parasolid kernel.

Note: For the 64-bit version of Parasolid under SPARC Solaris, this directory is called `base_sun64`. For the 64-bit version of Parasolid under HP-UX, this directory is called `base_hp64`. For the Pentium 4 version of Parasolid, this directory is called `base_p4`.

The top-level directory contains the following files.

File	Contents
fg.c	Foreign Geometry source code
fg.lib	Foreign Geometry library
fg_library.bat, fg_library.com	Commands to compile 'fg.c' and create a Foreign Geometry library
frustrum.c	Dummy frustrum source code
frustrum_delta.c	Example Partitioned Rollback Frustrum code
frustrum.lib	Dummy frustrum library
frustrum_ifails.h	Return failure codes include file for Frustrum
frustrum_library.bat, frustrum_library.com	Commands to compile 'frustrum.c' and create Frustrum library
frustrum_link.bat, frustrum_link.com	Commands to compile and link 'frustrum_test'
frustrum_test.c	Source of Frustrum library acceptance test
frustrum_tokens.h	Token names include file for Frustrum
kernel_interface.h	KI header file
kid_test.lsp	KID acceptance test
pskernel_archive.lib	Parasolid library. Formerly `parasolid.lib`.
parasolid_debug.h	Unsupported PK functions that aid application debugging
parasolid_ifails.h	Return failure codes include file for Parasolid
parasolid_kernel.h	PK interface header file for functions available in full Parasolid kernel.
parasolid_link.bat, parasolid_link.com	Commands to compile and link 'parasolid_test.c'
parasolid_test.c	Source of Parasolid library acceptance test
parasolid_tokens.h	Token names include file

File	Contents
parasolid_typedefs.h	Supporting typedefs for KI
testfr.lib	Library containing the Frustrum test function

The ‘lispdata’ sub-directory

This directory contains just one file.

bbcini.lsp	KID LISP initialization start-up file

The ‘schema’ sub-directory

This directory contains a number of schema files whose names have the form “sch_ ... .sch_txt”.

The ‘dll’ sub-directory (NT)

This directory contains a Parasolid DLL and its corresponding import libraries, and a KID executable linked to the full Parasolid kernel as a shared image. The following DLL is available:

- `pskernel.dll` – standard DLL for full Parasolid kernel

The ‘shared_object’ sub-directory (UNIX)

This directory contains the Parasolid library and KID as shared images. The shared object files available are as follows:

- `libpskernel.so` – standard file for full Parasolid kernel
- `libpskernel.sl` – standard file for full Parasolid kernel (HP only)

4.1.3 The ‘base_alternative’ directory

The `base_alternative` directory contains all the files needed to install and use Parasolid Communicator. The top-level directory contains the following files.

File	Contents
frustrum_ifails.h	Return failure codes include file for Frustrum
frustrum_tokens.h	Token names include file for Frustrum
parasolid_communicator.h	PK interface header file for functions available in Parasolid Communicator.
parasolid_ifails.h	Return failure codes include file for Parasolid
parasolid_tokens.h	Token names include file

The ‘schema’ sub-directory

This directory contains a number of schema files whose names have the form “sch_ ... .sch_txt”.

The ‘dll’ sub-directory (NT)

This directory contains the DLL for Parasolid Communicator (`pscommunicator.dll`) and its corresponding import library:

4.2 Explanation of files

4.2.1 Files in ‘base’

Parasolid library

This is the library of modeling subroutines which will be accessed by the application program via the PK interface functions.

PK and KI interface header files

Interface functions and type declarations for C applications.

Ifails and tokens include files

These are necessary for an application program to avoid using numeric values for PK arguments. They are specific to C, but can be easily edited, for example, to be FORTRAN integer variables.

Parasolid test code

This is included so that you can make a simple test that the library is set up correctly before integrating it with the application program. Parasolid requires a Frustrum - thus a dummy frustrum library is also included. This should only be used for acceptance testing, any customer Frustrum should be written to suit the application. Parasolid also requires Foreign Geometry functions - example functions are included.

Frustrum test code

This tests the dummy frustrum library in isolation from the Parasolid library. It uses the Frustrum tester function (TESTFR), see the Parasolid Downward Interface manual for a full explanation of this. This program can also be used to test your Frustrum.

Frustrum source code and library

Supplied for running the above.

Frustrum library command file

Supplied to build a Frustrum library from the source code.

Frustrum link and Parasolid link command files

These are supplied to link the test programs with the libraries. They can be copied and modified to link any program with Parasolid.

4.2.2 The Kernel Interface Driver (KID)

The KID program provides a LISP front-end to the Parasolid library, which can be used to learn the Kernel, prototype applications and report faults. For a full description see the Parasolid Kernel Interface Driver (KID) Manual. KID loads the LISP initialization file at start-up. The KID test file is supplied as an acceptance test for the KID program.

4.2.3 Schema files

Schema files describe the format and content of Parasolid part files (transmit or partition files), and they are supplied so that a Parasolid-powered application can

- read part files created using other versions of Parasolid
- write part files that can be read in older versions of Parasolid

A schema file is a platform independent, text based file with a name of the form `sch_`*`XXX`*`.sch_txt` (or `sch_`*`XXX`*`.s_t` for DOS FAT filesystems).

Each major version (e.g. V11.0, V12.0) has its own schema file. Minor versions (e.g. V11.1) usually use the most recent schema file. The appropriate schema file must be present when transmitting or receiving a part file. For instance if you are trying to read a V11.0 or V11.1 part file into Parasolid V12.0, the relevant schema file (`sch_11004`, in this case) must be present.

Parasolid guarantees upward compatibility of part files between major versions of Parasolid. To support this, schema files corresponding to all previous versions of Parasolid are provided with each new release.

Parasolid also guarantees two-way compatibility within a major version of Parasolid (e.g. a V11.0 file can be read into V11.1 and vice-versa).

Downward compatibility of Parasolid files between major versions is not supported. For example, you cannot read a v12.0 file into v11.1.

Warning: You should not attempt to “force” downward compatibility by copying the schema file for a later release into an earlier release of Parasolid. The behavior under these circumstances is not guaranteed or supported by Parasolid. Furthermore, the behavior of downstream operations on such parts is also not guaranteed.

4.2.4 Finding shared libraries

Customers can link the shared Parasolid image into the application and can then do upgrades/patches by just providing a new image. The customer must make sure that the Parasolid version of the new image is compatible with the old one.

To check this at run time, the application writer can use run-time library functions to open the shared image (e.g. dlopen() on some UNIX machines) and call PK_SESSION_ask_kernel_version to extract the version information.

The shared KID image (available only with the full Parasolid kernel) contains embedded information about where to look for the Parasolid library. This information may need to be overridden, so that the run-time loader will search the directory where the customer has put the libraries:

UNIX

There is usually an environment variable (often called LD_LIBRARY_PATH) that can be used to override the library search paths built into the image.

NT

The PATH environment variable is also used as the DLL search path.

Using Parasolid 5

5.1 Parasolid

UNIX Parasolid is supplied as

- **A library.** An executable image can be made by linking the object of a main program with the Parasolid, Frustrum and Foreign Geometry libraries; and
- **A shared object library.** An executable image can be made by linking the object of a main program with the Frustrum and Foreign Geometry libraries and the Parasolid shared object library in the 'shared_object' sub-directory.

In order to use the shared object library supplied, applications must register a frustrum using PK_SESSION_register_frustrum – see Chapter 5, "Registering the Frustrum" in the *Downward Interfaces* Manual for further details.

NT Two versions of Parasolid can be used in your applications, as follows:

- The main Parasolid kernel, containing the whole of Parasolid.
- Parasolid Communicator, a cut-down version of Parasolid containing limited functionality, for use in smaller applications that do not require access to the full range of Parasolid functionality.

The main Parasolid kernel is supplied as:

- **An object file library** (`pskernel_archive.lib`) which has been created using the Microsoft Visual C/C++ 32 bit compiler. An executable image can be made by linking the object of a main program with the Parasolid, Frustrum and (if required) Foreign Geometry libraries.
- **A DLL** (`pskernel.dll`) created from the object file library (using all default options with the exception of `/BASE:0x40000000`). An executable image can be made by linking the object of a main program with the Frustrum and (if required) Foreign Geometry libraries and the Parasolid 'interface library', `pskernel.lib` in the `dll` directory.

Function headers for the functions available in the full Parasolid kernel can be found in the file `parasolid_kernel.h`.

Parasolid Communicator is supplied as:

- **A DLL** (`pscommunicator.dll`) created from the object file library (using all default options with the exception of `/BASE:0x40000000`). An executable image can be made by linking the object of a main program with the Frustrum and (if required) Foreign Geometry libraries and the Parasolid 'interface library', `pscommunicator.lib` in the `dll` directory.

Function headers for the functions available in the Parasolid Communicator can be found in the file `parasolid_communicator.h`.

In order to use the supplied DLLs, your application must register a frustrum using PK_SESSION_register_frustrum – see Chapter 5, “Registering the Frustrum” in the *Downward Interfaces* manual for further details. See Chapter 8, “Environment”, in this manual, for more details on building and linking Parasolid.

The file `parasolid_link.com`, or `parasolid_link.bat` on NT platforms, shows examples of linking or binding Parasolid.

Note: For applications not using the Frustrum registration mechanism, Foreign Geometry functions must be linked in - even if the Foreign Geometry functionality is not used - to resolve all ‘downward’ references from Parasolid. A file, `fg.c`, of example functions is supplied for this purpose.

5.2 Schema Files

When transmitting part files, the schema file for the current version of Parasolid will be written to the schema directory of your Parasolid installation if it is not already present. The schema directory must be writable for this reason. If a previous version of Parasolid has been used to transmit parts which are to be received into a new version then the schema file for the previous version must be present in the schema directory of the new version.

In an NT application, your frustrum can look for schema files in one of two ways:

- Use a registry setting
- Use an environment variable

The dummy frustrum uses an environment variable called P_SCHEMA.

Note: If you use the dummy frustrum on NT, the %P_SCHEMA% environment variable is assumed to point to a directory on an NTFS file system. If %P_SCHEMA% points to a DOS FAT file system instead, you must copy and rename the schema files so that they have the extension `.s_t`.

Take care to preserve schema files when updating Parasolid versions.

5.3 Frustrum

When writing simple Parasolid programs, you may wish to initially use the dummy frustrum provided. The command files ‘frustrum_library.com’ or

'frustrum_library.bat' on NT platforms will create you a dummy frustrum library from the 'C' file, although this library is provided with the release.

The dummy frustrum source is provided primarily as a guide to help you develop your own frustrum. It also gives you a frustrum to use during the initial phases of the application development.

Note: On NT and Windows 2000 we recommend that your frustrum creates text files as stream, i.e. LF terminated, rather than using the DOS default (CR and LF terminated). Implementation details can be found in the dummy frustrum.

The dummy frustrum is not fully functional and does not contain all of the system calls required for a working frustrum. See the notes in the source file, 'frustrum.c' for further details.

If you wish to use the dummy frustrum, then you must either set an environment variable called P_SCHEMA to be able to access the schema files, or you must modify the code in the dummy frustrum so that the schema directory is specified explicitly, and P_SCHEMA is not used.

If you decide to set P_SCHEMA rather than modify the code, do it as follows:

UNIX

Bourne shell `$ P_SCHEMA="full SCHEMA directory pathname"`

'C' shell `$ setenv P_SCHEMA "full schema directory pathname"`

NT

`P_SCHEMA=full SCHEMA directory pathname`

Your finished application should not use the dummy frustrum.

You will need to make sure this command is executed each time you log in. The simplest way to do this is to specify it in the Environment tab of the System Control Panel applet. Alternatively, you can add the statement to your '.profile' script or 'login.com' file as appropriate. The full pathname needs to be given so that Parasolid can be run from any directory location.

Note: Unlike UNIX, you do *not* explicitly use quote (") marks when specifying environment variables in NT. This is true even if the pathname contains white space characters.

5.4 Kernel Interface Driver (KID)

5.4.1 Specifying environment variables

In order to run KID, you need to specify some environmental variables :

- PARASOLID refers to the ‘base’ directory; it provides a useful shortcut to specifying the full pathname to this directory at the command line.
- P_LISP points to a directory that contains information needed by KID.
- P_SCHEMA points to the directory that contains the schema files.

If your application uses the dummy frustrum, then you may have already specified P_SCHEMA.

UNIX

The environment variables should be set up as follows:

Bourne shell

```
$ PARASOLID="base directory pathname"
$ P_LISP="full LISP directory pathname"
$ P_SCHEMA="full SCHEMA directory pathname"
$ export PARASOLID P_LISP P_SCHEMA
```

‘C’ shell

```
$ setenv PARASOLID "base directory pathname"
$ setenv P_LISP "full LISP directory pathname"
$ setenv P_SCHEMA "full schema directory pathname"
```

NT

The environment variables should be set up as follows:

```
PARASOLID=base directory of pathname
P_LISP=full LISPDATA directory pathname
P_SCHEMA=full schema directory pathname
```

You need to make sure these commands are executed each time you log in. The simplest way to do this is to specify them in the Environment tab of the System Control Panel applet. Alternatively, you can add the statements to your ‘.profile’ script or ‘login.com’ file as appropriate. The full pathnames should be given as this will permit Parasolid to be run from any directory location.

5.4.2 Running KID

KID is supplied as an executable file and can be invoked as described below

UNIX Type the following at a UNIX prompt:

```
$ $PARASOLID/kid.exe
```

When executed, the image will attempt to load the file ‘$P_LISP/bbcini.lsp’

To allow file names of 255 characters (instead of the 14 character default length specified by some versions of UNIX) it is recommended that the UNIX operating system is re-configured.

If this is not done, KID will be limited to working with six character file keys and problems may occur when using rollback.

NT Either open the base directory in Windows Explorer and double-click the file kid.exe, or type the following in a command prompt window:

```
$ %PARASOLID%\kid.exe
```

When executed the image will attempt to load the file '%P_LISP%\bbcini.lsp'.

5.4.3 KID Graphical Device Drivers

Opening a Graphics Window

HPPA HPUX An X-window graphics driver is included within KID. To use this driver, invoke X-windows before executing KID, then before using any other graphics commands, type at the LISP prompt:

```
> (graphics open_device 'x)
```

KID creates a window, like any X-window, that can be re-sized, moved, etc. However you will need to refresh the graphics each time the window is manipulated by using:

```
> (graphics redraw)
```

SPARC Solaris KID supports graphics on the SPARCstation via OpenWindows. An X-window graphics driver can be used by invoking OpenWindows before starting KID, then typing the following at the LISP prompt, before using any other graphics commands:

```
> (graphics open_device 'x)
```

NT A Windows NT graphics driver is included within KID. To use this driver, type:

```
> (graphics open_device 'nt)
```

A separate graphics window will open; note that it isn't automatically placed in the foreground, and thus can be hidden by your console/lisp window.

On all other platforms: The KID image provides graphics support for X-windows. To use this, type the following at the LISP prompt, before using any other graphics commands:

```
> (graphics open_device 'x)
```

Null graphics device

If no graphics are required, use:

```
> (graphics open_device 'null)
```

Opening a null device will allow you to call Parasolid rendering functions without getting graphics output (for example, for testing purposes).

Graphics Enquiry

You can ensure KID graphics are set up correctly using:

```
> (graphics enquire)
```

Functional Enhancements 6

6.1 Functional enhancements since V12.1

This chapter describes the functional enhancements that have been added to the current release of Parasolid.

Note: For detailed information on interface changes see Chapter 7, “Interface Changes”.

6.1.1 Identifying blend faces

You can now identify any constant radius blend faces in a supplied array of faces, together with their radii and convexities.

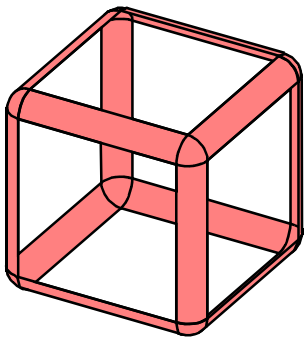

Figure 6–1 Identifying constant radius blends in a set of faces

For more information, see Chapter 40, “Simplifying Models”, of the *Functional Description*.

6.1.2 Identifying facesets

You can now divide a given body into facesets bounded by a set of supplied edges. A variety of options are available to let you return all facesets, only those that contain a set of supplied topologies, only those that exclude topologies, or alternating facesets.

For more information, see Chapter 40, “Simplifying Models”, of the *Functional Description*.

6.1.3 Identifying and deleting details in bodies

New functionality has been added that you can use to aid simplification of bodies.

It is now possible to return facesets in a body that are recognized as particular types of detail (e.g. holes), and you can delete specifed facesets from a given body. Used in conjunction, it is now possible to identify and remove particular details from a body in a robust manner.

For more information, see Chapter 40, “Simplifying Models”, of the *Functional Description*.

6.1.4 Double-sided tapering

Parasolid’s tapering functionality has been extended to provide single and double-sided tapering of a body. This specialized functionality lets you design parted molds in a single step.

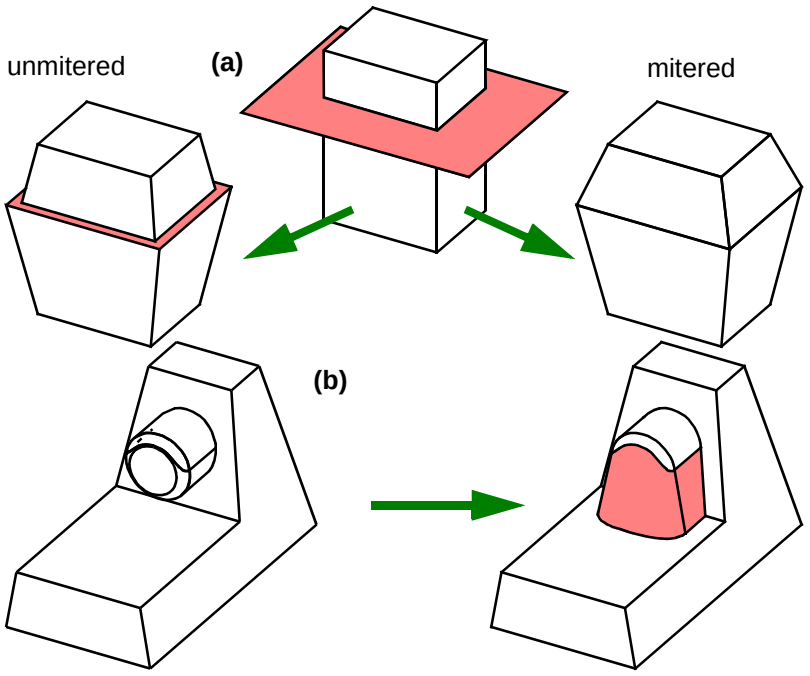

Figure 6–2 Performing (a) double-sided and (b) single-sided tapering on a body

For more information, see Chapter 21, “Local Ops: Double-Sided Tapering”, of the *Functional Description*.

6.1.5 Advanced capping in edge blending

Previously, it was not possible to cap an edge blend if the edges that were extended to cap the blend intersected. This restriction has now been removed.

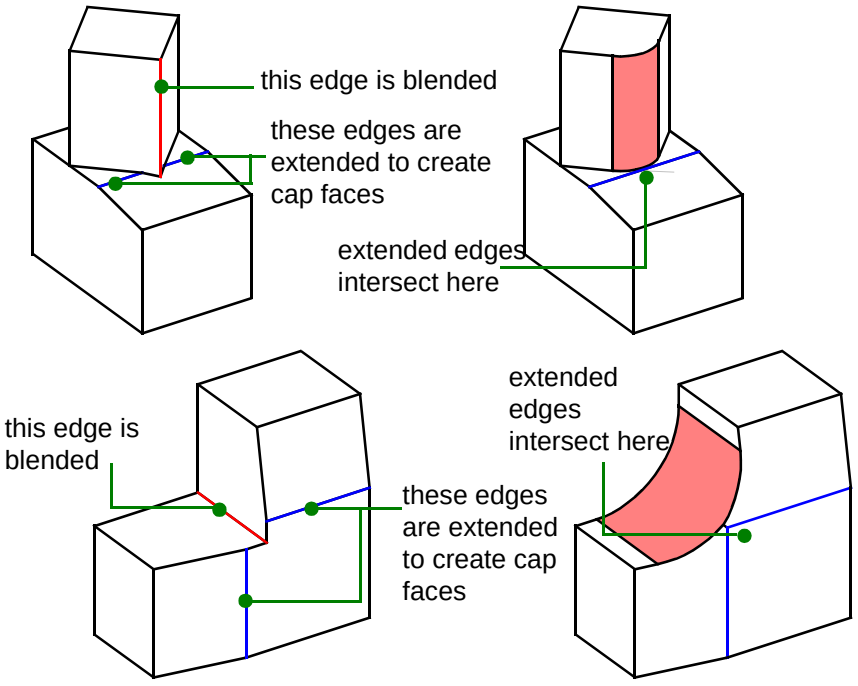

Figure 6–3 Capping edge blends where edges extended to form cap intersect

This functionality can be switched off using the `update` control in PK_BODY_fix_blends. For more information, see Chapter 30, “Edge Blending Functions and Options”, of the *Functional Description*.

6.1.6 Generating patches for filling holes in sheets

Parasolid can now generate faces to fill a hole in a sheet body. Previously your application was required to supply a suitable sheet body. When Parasolid generates a patch, the boundary of the hole is maintained, and the filling geometry meets the surrounding faces on the hole smoothly.

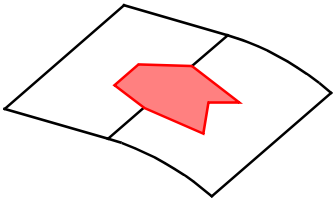

Figure 6–4 Generating a patch to fill a hole

For more information, see Chapter 16, “Local Ops: Creating Surfaces to Attach to Faces”, of the *Functional Description*.

6.1.7 Applying derivative constraints during lofting

Derivative constraints can now be applied to guide wires and intermediate profiles when constructing a lofted body. Previously, derivative constraints could only be applied to loft profiles.

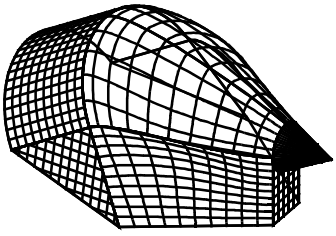

Figure 6–5 Applying derivative constraints to guide wires during lofting

For more information, see Chapter 27, “Advanced Surfacing”, of the *Functional Description*.

6.1.8 Improvements to Y-blend functionality

Parasolid’s functionality for creating Y-shaped blends has been extended. You can now create two Y-shaped blends that meet at a vertex. Any intermediate edges between the blends must be smooth for the blends to be successful. Additionally, you can now apply a Y shaped blend in cases where two edges forming a Y-blend do not meet smoothly, providing that there is a second Y-blend at the same vertex.

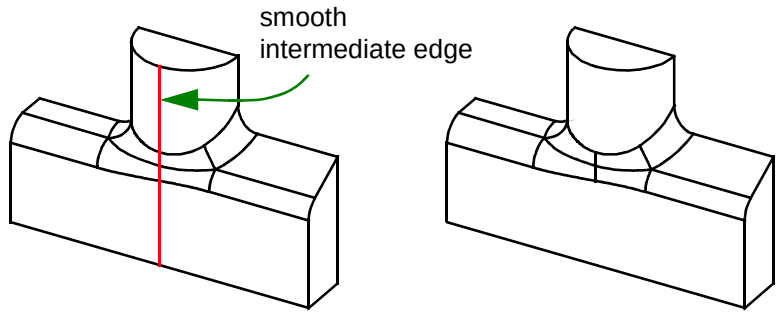

Figure 6–6 Creating two Y-shaped blends at a vertex

For more information, see Chapter 30, “Edge Blending Functions and Options”, of the *Functional Description*.

6.1.9 Improvements to face-face blending

Parasolid’s face-face blending functionality has been improved as follows:

- You can now specify the cross-sectional shape of a face-face blend directly.
- Face-face blending now supports the creation of chamfer blends, in addition to curvature continuous and conic blends.
- You can now create constant radius blends with disc and isoparameter cross-section planes. Previously, these were only possible with rolling-ball cross-section planes.

For more information, see Chapter 33, “Face-Face Blending”, of the *Functional Description*.

6.1.10 Specifying blend radius during face change operations

When regenerating blends as part of a face change operation, you can now specify the radius of the blend when it is reapplied. If no blend radius is specified then the blend radius of the original blend is used when the blend is reapplied.

For more information, see Chapter 23, “Local Ops: Generic Face Editing”, of the *Functional Description*.

6.1.11 Enhanced support when replacing the surfaces of faces

Parasolid has enhanced its support for replacing the surfaces of faces.

You can now replace the surfaces of several faces with a single replacement surface by passing the replacement surface to Parasolid once. Previously, you needed to pass the replacement surface several times: once for each face being replaced.

You can now control how adjacent faces are merged after replace face operations, in cases where the faces become mergable, as shown in Figure 6–7.

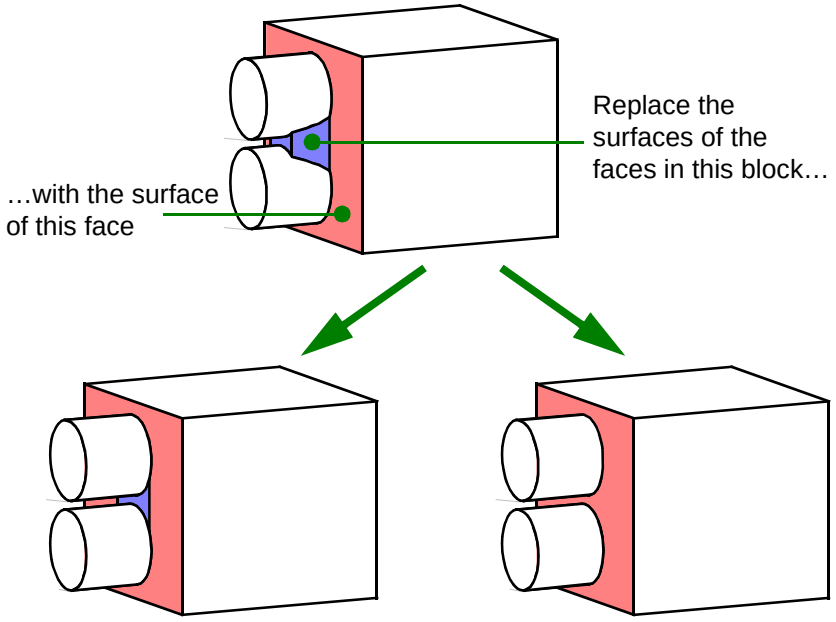

Figure 6–7 Controlling merging of faces during replace face operations

No merging is performed for edges for which replacement curves have been provided. Merging is recommended when adjacent faces are to acquire identical surfaces.

For more information, see Chapter 17, “Local Ops: Tweaking the Surfaces of Faces”, and Chapter 23, “Local Ops: Generic Face Editing”, of the *Functional Description*.

6.1.12 Creating step offsets

Parasolid can now automatically create offset step faces along smooth boundaries between offset and non-offset faces in a body.

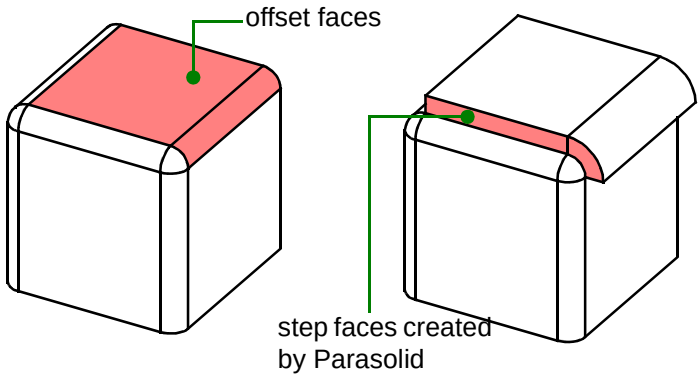

Figure 6–8 Creating step faces during offsetting

For more information, see Chapter 19, “Local Ops: Hollowing & Offsetting”, of the *Functional Description*.

6.1.13 Improvements to sweep

When sweeping, you can now remove undesirable twists from the swept body. This is done by ignoring regions of the specified path with rapidly changing curvature.

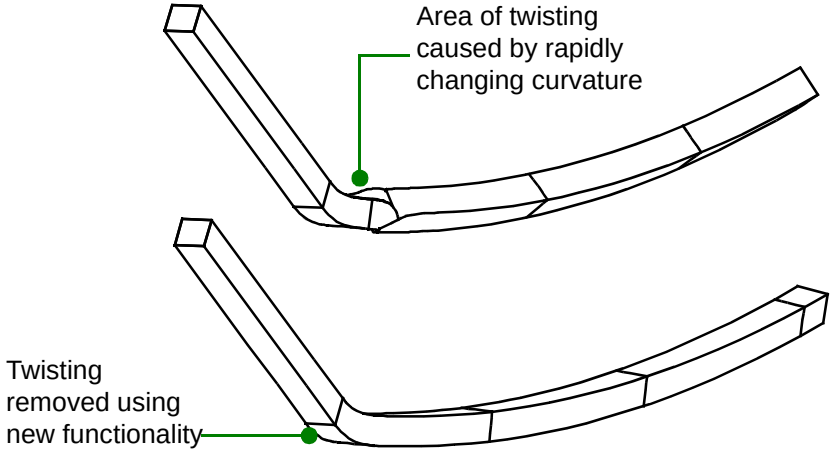

Figure 6–9 Removing undesirable twisting in sweep operations

For more information, see Chapter 27, “Advanced Surfacing”, of the *Functional Description*.

6.1.14 Improvements to clash detection

Detection of clashing topology has been improved. Previously, it was only possible to detect clashes between two sets of bodies. It is now possible to detect clashes between bodies, faces, or both.

- When two bodies, or a body and a face, are clashed, a list of clashing faces is returned
- When two faces are clashed, a list of clashing edges is returned.

For more information, see Chapter 9, “Boolean Operations”, of the *Functional Description*.

6.1.15 Simplifying geometry

You can now simplify the B-geometry for a supplied list of faces and their edges. Previously, it was only possible to simplify the B-geometry of an entire body.

For more information, see Chapter 40, “Simplifying Models”, of the *Functional Description*.

6.1.16 Removing duplicate sheets when sewing

When sewing sheet bodies together, you can now request that sheets are removed if they are duplicates to within the specified gap width bound of the sewing operation.

For more information, see Chapter 36, “Sheet Sewing”, of the *Functional Description*.

6.1.17 Smoothness tolerance between rendered faces

When rendering edges, you can now specify an angle of tolerance, within which faces on either side of the rendered edge are considered smooth.

For more information, see Chapter 53, “Rendering Option Settings”, of the *Functional Description*.

6.1.18 Control of coincident edges in booleans or sectioning

You can now control which of two coincident edges survives in a boolean or section operation.

For more information, see Chapter 9, “Boolean Operations”, and Chapter 11, “Sectioning”, of the *Functional Description*.

6.1.19 Attaching curve copies when sharing not possible

Previously, curves that were already attached to edges, fins or nominal geometry, were shared whenever possible. Attaching the curve failed if this sharing was not possible. This behavour has now changed, so that if sharing is not possible a copy of the curve is made and attached to the edge.

6.1.20 Creating bodies with curves in different partitions

When creating wire bodies using curves from several different partitions, any input curves in a partition other than the current partition are duplicated. The resultant body is in the current partition.

6.1.21 Multiple viewports

You can now use multiple viewports with PK_TOPOL_render_line. The existing single viewport functionality remains unchanged. An extra option is available to let you specify an array of multiple viewports.

For more information, see Chapter 53, “Rendering Option Settings”, of the *Functional Description*.

6.1.22 Parameters for facet vertices and degeneracies

When a facet vertex occurs at a degeneracy, the value of the degenerate parameter at the degenerate vertex can now be calculated from the average of the values of that parameter at the two facet vertices at the other ends of the facet edges which meet at the degenerate vertex. The normals and derivatives at the degenerate vertex are evaluated using this average value for the degenerate parameter.

For more information, see Chapter 54, “Facet Mesh Generation”, of the *Functional Description*.

6.1.23 Adding change events to bodies to the bulletin board

You can now specify that change events to bodies should appear on the bulletin board. If this is specified, then the first change to a body after the bulletin board was created or emptied results in a change event on the bulletin board.

For more information, see Chapter 57, “Bulletin Board”, of the *Functional Description*.

6.1.24 Support for Unicode part keys

Parasolid now supports part keys in 16-bit Unicode form as well as the native character set. To preserve runtime compatibility with older applications, the extra fields in various Frustrum structures must be explicitly enabled via PK_SESSION_set_unicode; otherwise the fields are ignored and the appropriate receive/transmit functions will fail.

For more information, see the *Downward Interfaces* manual, in particular, Chapter 2, “File Handling”, Appendix A, “Frustrum Functions”, and Appendix E, “Application I/O Functions”.

Note: Your application code must be fully re-compiled if you wish to make use of Unicode functionality.

6.1.25 Support for transmitting parts in XML format

To support the Parasolid eXT schemas, you can now transmit text versions of parts, partitions, and sessions with character data translated into a format suitable for embedding in XML files. This format escapes the “<“ and “&” characters.

For more information, see Chapter 42, “Archives”, of the *Functional Description*.

6.1.26 Setting the amount of memory requested by Parasolid

You can now configure and enquire about the minimum amount of contiguous memory space that Parasolid requests for modeling operations. This can be done at any point – even when the modeler is not running. You can also reset the default smallest block size at any time. When the modeler is stopped the smallest block size is always returned to Parasolid’s default block size.

For more information, see Chapter 58, “Session Support”, of the *Functional Description*.

6.2 Known issues in this release

This section includes miscellaneous issues that are not covered elsewhere in this chapter.

6.2.1 Pentium 4 issues

The Pentium 4 version of Parasolid V13.0 FCS was built using a beta-release compiler, and is therefore a beta release. A version built using a production-release compiler will be made available in a later patch release.

SMP is currently switched off in the Pentium 4 version of Parasolid V13.0.

6.2.2 HP-UX 64-bit issues

SMP is currently switched off in the 64-bit version of Parasolid V13.0.

Interface Changes 7

7.1 Introduction

This chapter describes changes that have been made to the PK interface for Parasolid V13.0. This information is structured as follows:

- Section 7.2 describes changes to existing PK functions and typedefs.
- Section 7.3 describes new PK functions and typedefs that have been added in this release.
- Section 7.4 lists new PK interface tokens
- Section 7.5 lists new PK interface error codes
- Section 7.6 describes changes to Parasolid that are not documented, and should not therefore be relied on in your application code
- Section 7.7 lists significant changes to the PK header documentation other than those changes that have been necessary as a result of new or changed functionality
- Section 7.8 lists undocumented functions that should not be used in your code.

7.2 PK functions and typedefs with changes in behavior

This section describes enhancements made to existing PK functions and typedefs since Parasolid V12.1, resulting in a change to the function interface or a change in behavior.

We recommend that you review the use of these functions in your applications.

7.2.1 Specifying blend radius during face change operations

Changed structure PK_FACE_change_blend_o_t

If the operation PK_FACE_change_type_blend_c is requested in PK_FACE_change, you can now specify the radius of the blend when it is reapplied. If no blend radius is specified the radius used when the blend is reapplied is that of the original blend.

7.2.2 Errors during face change operations

Changed function PK_FACE_change

PK_FACE_change can now return the errors

- PK_ERROR_failed_to_blend
- PK_ERROR_failed_to_offset
- PK_ERROR_failed_to_taper
- PK_ERROR_failed_to_replace
- PK_ERROR_failed_to_transform

If such an error is returned at least one of the requested operations has failed.

7.2.3 Changes to blending algorithms

New structure PK_blend_edge_update_t

Changed structure PK_BODY_fix_blends_o_t

The new `update` option enables or disables some changes to the blending algorithms that might cause differences in model updates between different versions of Parasolid.

7.2.4 Errors during outline curves operations

Changed function PK_BODY_make_curves_outline

PK_BODY_make_curves_outline may return:

- PK_ERROR_unsuitable_entity if the given entities are not solid or sheet bodies
- PK_ERROR_non_manifold if the result would not be a set of closed loops
- PK_ERROR_failed_to_make_outline if the function fails

7.2.5 Invalid matched regions

Changed functions PK_BODY_boolean
PK_BODY_boolean_2

These functions now return PK_ERROR_invalid_match_region if the topologies in the matched region are not in the target or tools for the boolean operation.

7.2.6 Control of coincident edges in booleans or sectioning

Changed structures PK_BODY_boolean_o_t
PK_FACE_boolean_o_t

PK_BODY_section_o_t
PK_FACE_section_o_t

There is a new `keep_target_edges` option that controls which edge survives when two edges are coincident in a boolean or section operation.

7.2.7 Clashing faces

Changed function PK_TOPOL_clash

PK_TOPOL_clash has been modified to allow individual faces to be clashed together. Previously, it was only possible to clash entire bodies.

- When two bodies, or a body and a face, are clashed, a list of clashing faces is returned
- When two faces are clashed, a list of clashing edges is returned.

7.2.8 Attaching curve copies when sharing not possible

Changed function PK_EDGE_attach_curve_nmnl

Previously, PK_EDGE_attach_curve_nmnl shared curves that were already attached to edges, fins or nominal geometry, and failed if sharing was not possible. This behavour has now changed, so that if sharing is not possible a copy of the curve is made and attached to the edge.

7.2.9 Creating bodies with curves in different partitions

Changed function PK_CURVE_make_wire_body_2

When making a body with PK_CURVE_make_wire_body_2, any input curves in a partition other than the current partition will be duplicated. The resultant body will always be in the current partition.

7.2.10 Creating periodic B-curves

Changed structure PK_knot_type_t

A new value, PK_knot_do_wraparound_c, allows you to create periodic B-curves using the same data as in the KI function CRBSPC.

7.2.11 Removing duplicate sheets when sewing

Changed function PK_BODY_sew_bodies

Changed structure PK_BODY_sewing_removal_t

Identification and removal of duplicate sheets within PK_BODY_sew_bodies has been extended. You can now use the `duplicate_removal` option to request that sheets are removed if they are duplicates to within the tolerance given as the `gap_width_bound` of the sewing operation.

7.2.12 Parameters for facet vertices and degeneracies

Changed structures PK_facet_degen_t
PK_TOPOL_facet_mesh_o_t

PK_facet_degen_t can now take the value PK_facet_degen_average_parms_c, which allows multiple vertices to be output at singularities in the same way as PK_facet_degen_multiple_vxs_c.

In this case when a facet vertex occurs at a degeneracy, the value of the degenerate parameter at the degenerate vertex is the average of the values of that parameter at the two facet vertices at the other ends of the facet edges which meet at the degenerate vertex. The normals and derivatives at the degenerate vertex are evaluated using this average value for the degenerate parameter.

7.2.13 Adding change events to bodies to the bulletin board

Changed structure PK_BB_sf_t

You can now specify that change events to bodies should appear on the bulletin board. If this is specified in the PK_BB_sf_t argument to PK_BB_create, then the first change to a body after the bulletin board was created or emptied, will result in a change event on the bulletin board.

7.3 New Parasolid functionality

This section describes new PK functions and typedefs that have been added to Parasolid V13.0. These interface changes have been made to provide access to new functionality in Parasolid.

7.3.1 Improvements to face-face blending

Changed function PK_FACE_make_blend

Changed structures PK_FACE_make_blend_o_t
PK_blend_shape_t

New structures PK_blend_xs_shape_t
PK_blend_xs_plane_t

The capabilities of face-face blending have been expanded so that you can now choose the cross-sectional shape of any blend directly.

Any face-face blend may now be a chamfer, a curvature continuous blend or a conic blend. This is controlled by setting `xs_shape` in the PK_blend_shape_t substructure of the option structure.

You can now create constant radius blends with disc and isoparameter cross-section planes, using the `xsection` option. Previously, these were only possible with rolling-ball cross-section planes.

7.3.2 Identifying blend faces

New function PK_FACE_identify_blends

New structures PK_FACE_identify_blends_o_t
PK_FACE_identify_blends_r_f
PK_FACE_identify_blends_r_t
PK_comparison_t

It is now possible to identify any constant radius blend faces in a supplied array of faces, together with their radii and convexities.

7.3.3 Identifying facesets

New function: PK_BODY_find_facesets

New structures: PK_BODY_find_facesets_o_t
PK_BODY_find_facesets_r_t
PK_BODY_find_facesets_r_f

You can use PK_BODY_find_facesets to divide a body into a series of facesets bounded by a given array of edges.

7.3.4 Identifying and deleting model details

New functions: PK_BODY_identify_details
PK_FACE_delete_facesets

New structures: PK_BODY_identify_details_o_t
PK_FACE_delete_facesets_o_t
PK_detail_t
PK_identify_details_r_t
PK_identify_details_r_f

You can use PK_BODY_identify_details to return facesets in a body that correspond to specified types of detail. You can use PK_FACE_delete_facesets to remove as many of its given facesets as possible. These two functions can be used in conjunction to provide a robust method of identifying and removing certain types of detail from a body.

7.3.5 Double-sided tapering

New function PK_BODY_taper

New structure PK_BODY_taper_o_t

Parasolid's tapering functionality has been extended to provide single-pass double-sided tapering of a body. This function is particularly useful for mold design applications that implement parted casting design.

Note: It is not currently possible to omit a chain of references either above or below a parting line. This will be made possible in a future release of V13.

7.3.6 Creating step offsets

Changed structures PK_BODY_offset_o_t
PK_FACE_offset_o_t

A new option, `offset_step`, has been added to these options structures. This option creates step faces along any smooth boundaries between offset and non-offset faces when calling PK_BODY_offset_2 or PK_FACE_offset_2.

7.3.7 Controlling face merging when replacing faces

Changed structure PK_FACE_replace_surfs_o_t
PK_FACE_change_replace_o_t

New structure PK_replace_merge_t

It is now possible to replace the surfaces of multiple faces with a single surface by passing the replacement surface only once. Previously, it needed to be passed for every face being replaced.

A new datatype in PK_FACE_replace_surfs_o_t lets you control the required level of merging when adjacent faces become mergable as a result of a face replace operation. No merging is performed for edges for which replacement curves have been provided. Merging is recommended when adjacent faces are to acquire identical surfaces.

7.3.8 Generating patches for filling holes in sheets

Changed function PK_BODY_fill_hole

Changed structures PK_BODY_fill_hole_o_t
PK_fill_hole_fault_t
PK_fill_hole_method_t

Parasolid can now generate suitable faces to fill a hole using PK_BODY_fill_hole, rather than require that your application supplies a suitable sheet body. The boundary of the hole is maintained, and the filling geometry meets the surrounding faces on the hole smoothly where possible.

7.3.9 Improvements to sweep

Changed function PK_BODY_make_swept_body

Changed structure PK_BODY_make_swept_body_o_t

New structure PK_BODY_sweep_fair_t

A new PK_BODY_make_swept_body option has been introduced to allow internal improvement of “bad” sweep paths. This contains a single method that removes twists from regions of paths that have rapidly changing curvature.

7.3.10 Simplifying geometry

New function PK_FACE_simplify_geom

New structure PK_FACE_simplify_geom_o_t

You can now simplify the B-geometry for a supplied list of faces and their edges. Previously, it was only possible to simplify the B-geometry of an entire body.

7.3.11 Applying derivative constraints to guide wires

Changed structure PK_BODY_make_lofted_body_o_t

Derivative constraints can now be applied to guide wires when constructing a lofted body using PK_BODY_make_lofted_body.

7.3.12 Multiple viewports

Changed structures PK_TOPOL_render_line_o_t
PK_render_viewport_t

You can now use multiple viewports with PK_TOPOL_render_line. The existing single viewport functionality remains unchanged. An extra option is available to let you specify an array of multiple viewports.

7.3.13 Smoothness tolerance between rendered faces

Changed structure PK_TOPOL_render_line_o_t

You can now specify a tolerance on the smoothness between faces when rendering edges. This is specified using the new options `is_edge_smoothness_tol` and `edge_smoothness_tol` in PK_TOPOL_render_line_o_t. If the angle between the normals of the relevant faces is below `edge_smoothness_tol`, the edge is considered smooth and indicated as such.

7.3.14 Support for Unicode part keys

Changed functions PK_SESSION_ask_frustrum
PK_SESSION_register_frustrum
PK_SESSION_ask_applio
PK_SESSION_register_applio

New functions PK_SESSION_ask_unicode
PK_SESSION_set_unicode

Changed structures PK_UCOPRD_f_t
PK_UCOPWR_f_t
PK_SESSION_applio_t

Parasolid now supports part keys in 16-bit Unicode form as well as the native character set. To preserve runtime compatibility with older applications, the extra fields in various Frustrum structures must be explicitly enabled via

PK_SESSION_set_unicode; otherwise the fields are ignored and the appropriate receive/transmit functions will fail.

7.3.15 Support for transmitting parts in XML format

Changed structure: PK_transmit_format_t

To support embedding XT files in XML data, the new token PK_transmit_format_xml_c allows you to generate text part, partition, and session files in which the < and & characters are escaped with `<` and `&` respectively.

7.3.16 Setting the amount of memory requested by Parasolid

New functions PK_MEMORY_ask_block_size
PK_MEMORY_set_block_size

A new function, PK_MEMORY_set_block_size, has been added which allows you to set the smallest block of contiguous space that Parasolid requests when acquiring workspace and modeling memory via FMALLO. PK_MEMORY_set_block_size lets you specify a minimum block size between 130,776 bytes (the default) and 16,736,280 bytes. It can be called at any time, whether or not the modeler is running.

PK_MEMORY_ask_block_size returns the current minimum block size.

The minimum block size can be returned to Parasolid's default by calling PK_MEMORY_set_block_size with an argument of zero. When the modeler is stopped the smallest block size is always returned to Parasolid's default block size.

Note: If the smallest block size is increased above the default of 130,776 bytes, then the amount of memory permanently retained by Parasolid increases accordingly.

7.4 New PK Interface tokens

PK_knot_type_t

PK_knot_smooth_seam_c	8506

PK_boolean_region_t

PK_boolean_off_c	15949

PK_FACE_heal_t

PK_FACE_heal_yes_c	18085

PK_transmit_format_t

PK_transmit_format_xml_c	18224

PK_blend_xs_plane_t

PK_blend_xs_rolling_ball_c	18490
PK_blend_xs_disc_c	18491
PK_blend_xs_isoparameter_c	18492

PK_render_viewport_t

PK_render_viewport_array_c	20422

PK_facet_degen_t

PK_facet_degen_average_parms_c	20582

PK_local_status_t

PK_local_status_not_supported_c	21465

PK_fill_hole_fault_t

PK_fill_hole_not_smooth_c	22093
PK_fill_hole_too_complex_c	22094

PK_fill_hole_method_t

PK_fill_hole_create_patch_c	22102

PK_blend_edge_update_t

PK_blend_edge_update_0_c	22190
PK_blend_edge_update_1_c	22191

PK_blend_xs_shape_t

PK_blend_xs_shape_unset_c	22200
PK_blend_xs_shape_conic_c	22201
PK_blend_xs_shape_g2_c	22202
PK_blend_xs_shape_chamfer_c	22203

PK_BODY_sweep_fair_t

PK_BODY_sweep_fair_no_c	22210
PK_BODY_sweep_fair_twist_c	22211

PK_comparison_t

PK_comparison_always_c	22220
PK_comparison_less_c	22221
PK_comparison_equal_c	22222
PK_comparison_greater_c	22223
PK_comparison_between_c	22224
PK_comparison_outside_c	22225
PK_comparison_never_c	22226

PK_blend_identify_t

PK_blend_identify_within_c	22240
PK_blend_identify_exc_chain_c	22241
PK_blend_identify_inc_chain_c	22242
PK_blend_identify_max_chain_c	22243
PK_blend_identify_dependent_c	22244

PK_detail_t

PK_detail_any_c	22250
PK_detail_rubber_c	22251
PK_detail_hole_cyl_c	22252
PK_detail_hole_cyl_through_c	22253
PK_detail_hole_cyl_blind_c	22254
PK_detail_hole_cyl_closed_c	22255
PK_detail_blend_rb_const_r_c	22256

PK_replace_merge_t

PK_replace_merge_no_c	22300
PK_replace_merge_in_c	22301
PK_replace_merge_out_c	22302

PK_offset_step_t

PK_offset_step_no_c	22310
PK_offset_step_yes_c	22311

7.5 New PK interface error codes

PK_ERROR_duplicate_name	5135
PK_ERROR_failed_to_make_outline	5136
PK_ERROR_failed_to_blend	5137
PK_ERROR_failed_to_offset	5138
PK_ERROR_failed_to_taper	5139
PK_ERROR_failed_to_transform	5140
PK_ERROR_journalling_on	5141
PK_ERROR_dbg_rprt_not_stopped	5142
PK_ERROR_dbg_rprt_not_started	5143

7.6 Undocumented changes

This section describes changes that may be made to Parasolid at any given release, but are not explicitly documented.

Warning: The following are **not** guaranteed to be consistent between different versions of Parasolid, and may change between releases without notice. Your application code should not rely on this behavior.

- The order in which entities are returned from a given PK function
- The underlying representation of faces and edges

7.7 Changes to the Parasolid documentation

In addition to documentation covering the changes and additions to Parasolid functionality, the following changes have been made to the PK Interface documentation that do not affect any existing code.

7.7.1 New manuals in the documentation set

Three new manuals have been added to the documentation set for Parasolid V13.0. These are:

- *PS/Workshop User Guide* – a guide to using the PS/Workshop application supplied with Parasolid.
- *PS/Workshop Developer Guide* – a guide to writing your own plu-in modules for use with PS/Workshop
- *Using the Example Application* – a guide to the structure of the example Parasolid-powered application whose source code is supplied with Parasolid.

These manuals are all available from the home page of the Parasolid HTML documentation set, and are available in PDF format by following the **PDF files** link on the home page.

7.7.2 New face-face blending documentation

The documentation for Parasolid's face-face blending functionality has been completely rewritten. For more information, see Chapter 33, "Face-Face Blending", of the *Functional Description*.

7.7.3 Documentation for datatypes and structures

PK reference documentation is now available for all datatypes of the form PK_*CLASS*_t and PK_*CLASS*_array_t. This means you can now click on any Parasolid-typed function argument or option field in the HTML PK reference documentation to read more information.

7.8 Undocumented PK functions

7.8.1 Debug functionality

Parasolid includes a number of PK_DEBUG functions that you can use to obtain additional information when debugging calls to PK functionality. Documentation

for these functions is provided in the *PK Interface Programming Reference Manual*, but there is no description of functionality in the accompanying *Functional Description*.

Note: You should not include calls to any PK_DEBUG functions in a released product.

7.8.2 Approximate evaluations on geometry

The following functions are present within Parasolid to support applications with legacy code. We do not recommend the use of these functions and as such, the functions have not been included in the documentation.

- PK_BCURVE_eval_approx
- PK_BCURVE_set_approx
- PK_BCURVE_unset_approx
- PK_BSURF_eval_approx
- PK_BSURF_set_approx
- PK_BSURF_unset_approx
- PK_SPCURVE_eval_approx

To obtain the documentation or information regarding these functions, contact Parasolid Technical Support.

7.8.3 Other undocumented functions

The following function is undocumented and should not be used in application code. It is provided only for backward compatibility

- PK_BODY_make_swept_profiles

Environment 8

8.1 Introduction

This chapter contains information on the platform specific operating systems and compiler options used to build the Parasolid library.

Floating Point Underflow Traps

Warning: Because Parasolid relies on ‘quiet underflow of denormals to zero’ it is important that you do not, should the option be available, enable floating point underflow traps.

8.2 Intel NT

Parasolid was compiled, linked and tested on Windows NT version 4.0 with Service Pack 5.

It will run under Windows NT version 4.0 or Windows 2000 on an Intel 80486, Pentium, Pentium Pro, Pentium II, Pentium III, or AMD Athlon processor. At least 32 Mbytes of RAM are recommended.

Windows 95 and 98 are not officially supported, but Parasolid-based products have been successfully created for them.

The C compiler used was Microsoft Visual C++ version 6.0 with Service Pack 4, with the following options:

```
/MD /DWIN32 /Gs /QIfdiv- /O2 /Oy- /G6 /DNDEBUG
```

where:

`/O2`	Optimize for speed
`/Gs`	Disable stack checking code
`/MD`	Link with MSVCRT.LIB
`/Oy-`	Use a stack frame

/G6	Optimize for Pentium Pro/Pentium II/Pentium III. Parasolid will still run on a Pentium or 80486.
/QIfdiv	Prevents checking of parameters to floating point calculations for values that trigger the 'Pentium bug'. Early versions of the Pentium processor had a fault which produced incorrect results on some floating point calculations. Modern Pentiums and all Pentium Pros/II/IIIs are free of this error. Checking for the critical parameter values allows code to run on the obsolete processors, but reduces Parasolid performance significantly.

To improve performance, Parasolid for Intel has not been compiled with the /Op option (Improve floating-point consistency). Applications that use 64-bit precision (i.e. 80-bit real numbers) may therefore suffer from inconsistent results; therefore we recommend the use of 53-bit precision (64-bit real numbers).

Also for performance reasons, Parasolid uses Visual C++'s default structure packing (equivalent to the /Zp8 compiler option). Do not use the /Zp*n* compiler option with any value of *n* except 8; if wrongly-packed structures are passed to Parasolid, it will crash.

Note: See also Section 8.9, "Linking NT run-time libraries".

The Pentium 4 version of Parasolid was built with the Intel C/C++ compiler, version 5.0.1, using the following options:

```
/MD /DWIN32 /DNDEBUG /Gs /QIfdiv- /O2 /Oy- /G7 /QxW /Qfp_port-
```

The options that differ from the Microsoft-compiled version are:

Option	Description
/G7	Optimize for the Pentium 4.
/QxW	Compile for the Pentium 4 and compatible NetBurst™ processors. The resulting code will not run on older Intel or AMD processors.
/Qfp_port-	Modify handling of conversions between double and float. Parasolid does not perform these conversions, so dispensing with the default handling has a small performance benefit. You should not use this option in code which actually does such conversions.

Running this version of Parasolid requires the Intel math library DLL libmmd.dll.

Attempting to run this version of Parasolid on a Pentium III, AMD Athlon, or other processor that does not support the Pentium 4 instruction set will produce an Illegal Instruction exception. The Parasolid DLL has additional version information, allowing you to check that you are using the Pentium 4 version via Windows Explorer (right-click the DLL file and choose **Properties** from the shortcut menu).

You may find that your application requires more stack space when using the Parasolid DLL. You can use the `/stack` parameter with the Microsoft linker to increase the stack size from the default. Using a larger stack space does not affect performance on P3 or earlier processors.

Note: The FCS release of the Pentium 4 version of Parasolid was compiled using a beta-quality compiler, and is therefore a beta release.

8.3 Linux

Parasolid was compiled, linked and tested on Red Hat Linux 6.0.

Currently, there is no shared library under Linux: only an archive library is available.

The C compiler used was gcc version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release), with the following options:

```
-O -fPIC
```

where:

`-O`	optimization on
`-fPIC`	generate position-independent code

8.4 HPPA HPUX

Parasolid was made using the operating system HP-UX 11.0 and C compiler A.11.01. The compiler options used were:

```
+O3 +Olibcalls +Onolimit +Onomoveflops +Onoinitcheck +DA1.1
+DS2.0a +Oentrysched +Z -Ae +w2 +ESlit
```

where:

`+O<option>`	optimization options
`+DA1.1`	use PA_RISC 1.1 architecture instruction set

+DS2.0a	instruction scheduling optimized for PA2.0 architecture
+Z	produce position independent object code
+ESlit	place string literals in read-only memory
-Ae	Extended ANSI mode

Note: On HP: we recommend that the -z parameter is used when linking, so that null pointer accesses are detected.

The 64-bit version of Parasolid is built with the same options, substituting +DD64 – which tells the compiler to produce 64-bit code – for +DA1.1 +DS2.0a.

To use the 64-bit version of Parasolid, all parts of your application code must be compiled for 64-bit.

8.5 SPARC Solaris

Parasolid is supplied for both 32-bit and 64-bit versions of SPARC Solaris. Parasolid was made using the operating system Solaris 5.7. The C compiler used was Sun Workshop C 5.0 with the following options:

```
-xtarget=ultra -xdepend -x04 -xlibmil -K PIC -xstrconst
-D_POSIX_SOURCE -dalign -mt -Xa
```

For the 64-bit version, the following option is also used:

```
-xarch=v9
```

Note: The -xarch option must come *after* the -xtarget option.

The options used have the following meaning:

-xdepend	do dependency analysis and restructuring of loops
-x04	optimization level 4
-xlibmil	inline some library routines for performance
-K PIC	generate position-independent code
-xstrconst	put string literals in read-only memory
-dalign	assume double data aligned appropriately for double word load/store instructions
-mt	required for multi-threading code

-Xa	ANSI C plus K&R C compatibility extensions
-xarch	Setting this to v9 builds a 64-bit version, rather than a 32-bit verson.

The xtarget=ultra option generates code optimized for SPARC V8 machines and uses the timing properties of the UltraSPARC chip. This may mean that code will run slightly slower on the older SuperSPARC architecture. The Parasolid libraries and KID, supplied with this release, were generated with this option. These files will also run on pre- SPARC V8 machines, such as SPARC2, but with a significant degradation in performance.

The 64-bit version does not run on pre-UltraSPARC machines, and requires the 64-bit Solaris kernel, which only runs as the default on UltraSPARC II and later machines.

Note: To use this Parasolid library, all parts of your application code must be compiled for 64-bit.

8.6 AXP OSF

Parasolid was made using the operating system OSF1 4.0 (the original name of Tru64). The C compiler used was the Compaq C compiler V5.6-071 with the options:

```
-migrate -O2 -G 0 -FASTMATH
```

where:

-migrate	set default compiler options as for pre-v4.0 compiler
-O2	optimization level 2
-FASTMATH	use faster versions of some math library routines

8.7 RS6000 AIX

Parasolid was made on an IBM RS/6000 workstation under the AIX 4.3 operating system. The AIX compiler version 3.6.6.0 options used were:

```
-O -qlanglvl=ansi -qcheck=all:nobounds
-D_POSIX_SOURCE -qro -qtune=604
```

where:

`-O`	optimization on
`-qlanglvl=ansi`	ansi mode (not IBM RT compatibility mode)
`-qcheck`	trap run-time exceptions except array bound checking
`-D_POSIX_SOURCE`	use Posix standard headers
`-qro`	put string literals in read only memory
`-qtune=604`	optimize for Power PC 604 processor

The shared libraries provided are `.so` files, rather than old-style `.a` files. You may need to add the `-brtl` option to your link command line, to tell the linker to look for `.so` libraries.

8.8 R4000 IRIX

Parasolid was compiled on the IRIX 6.5 operating system, using version 721 of the IRIX C compiler. The compiler options used were:

```
-n32 -mips3 -O2 -KPIC -G 0 -LANG:vla=off
```

where:

`-n32`	use newer 32 bit ABI (Application Binary Interface)
`-mips3`	use MIPS R4000 instruction set
`-KPIC`	generate position-independent code
`-O2`	optimization level 2

If you have an earlier version of the IRIX C compiler, we recommend that you upgrade to version 720 or later for work with n32 Parasolid.

Note: All parts of a program must be compiled with the same ABI, i.e. the application code must also be compiled with -n32 to use this Parasolid library.

Parasolid is capable of using large amounts of virtual memory. This means that the default configuration normally supplied with a Silicon Graphics workstation, (which allows any process to take the whole of this allocated virtual memory), is dangerous. Therefore, use the systune utility to examine and adjust the system parameters `rlimit_data_cur` and `rlimit_stack_cur`, and possibly `rlimit_data_max` and `rlimit_stack_max`, to ensure that any given process cannot allocate all available virtual memory.

8.9 Linking NT run-time libraries

Microsoft Visual C++ provides several different versions of the C run-time library. Unfortunately, the library to be used has to be selected before any code is compiled, which restricts the freedom of library manufacturers and application developers somewhat.

Parasolid is compiled with the `/MD` option, to use the dynamically-loaded version of the C runtime library. It must be linked with the interface library `msvcrt.lib`, and the run-time library DLL, `msvcrt.dll` must be available at run time.

The recommended method of building an NT application with Parasolid is to compile the application code with `/MD`, and to link it with either `pskernel.lib` or `pscommunicator.lib` (depending on whether you want the full Parasolid DLL, or the cut-down Parasolid Communicator DLL) and `msvcrt.lib`. This will use the DLL versions of Parasolid and the C run-time library.

You can link Parasolid statically to your application if you prefer: compile with `/MD` and link with `pskernel_archive.lib` and `msvcrt.lib`. This produces a single very large executable (Parasolid is more than 10Mb), which will require `msvcrt.dll` to be available.

If your application absolutely *has* to be statically linked to the C run-time library, then you could compile it with `/ML` and link with either `pskernel.lib` or `pscommunicator.lib` (depending on whether you want the full Parasolid DLL, or the cut-down Parasolid Communicator DLL) and `libc.lib`. Your application will use the appropriate DLL version of Parasolid and the statically linked version of the C library. `msvcrt.dll` must still be available at run-time, since the Parasolid DLL will require it. While this method has worked in tests, we do *not* recommend it, since it involves having two copies of the C run-time library in use simultaneously. This wastes memory and could confuse the library code. So far, Parasolid's frustrum architecture seems to have allowed this problem to be evaded, but it is very poor practice to risk it.

It is not possible to link Parasolid statically to a LIBC-based application; this would require all of Parasolid to be compiled with `/ML`. Overriding the library to be used by Parasolid with the `/NODEFAULTLIBRARY` link parameter will not work; some of the functions in the C run-time library have different names in the static and dynamic versions. Linking with only LIBC produces missing symbols, while linking with both produces duplicate symbols.

It is inadvisable to statically link an application compiled with `/MDd` to Parasolid. This option tells the linker to use the debug version of the C run-time library. This is not required for normal debug compilations; it is only needed for source-level debugging of the run-time library itself. Under these circumstances, the link will work, but running the program will produce assertion failures and access violations. It seems to be necessary to compile an entire program with `/MDd`, if any of it is compiled that way. Statically linking debug-compiled code without

`/MDd` to Parasolid works without problems; using `/MDd` code with a DLL Parasolid also works.