A LON publication

CogStd.dll

Dynamically linked library

for use with Cogent Graphics

Version 1.33

Programmer's guide

Written by J.Romaya W.D.C.N.

12th August 2013

Changes v1.28 to v1.33

1/ There are no changes between v1.28 and v1.33. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.27 to v1.28

1/ .dll files renamed to .mex and .mexw32 files added for each .mex file. This allows compatibility with Matlab r2007b.

Changes v1.25 to v1.27

1/ There are no changes between v1.27 and v1.25. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.24 to v1.25

1/ A new function has been added; sDXVer(). This returns the DirectX version number currently installed on the PC multiplied by 100.

2/ A new function has been added; sOSID(). This returns the name of the operating system as an ascii string.

Changes v1.23 to v1.24

1/ A new function has been added; sPriority(). This sets the priority class of the current process. It should be used to improve the reliability of the gFlip() command with regard to dropped frames.

Changes v1.22 to v1.23

1/ Warning messages are now enabled by default and have to be explicitly turned off by a call to sEnable.

Changes v1.20 to v1.22

1/ There are no changes between v1.20 and v1.22. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.19 to v1.20

1/ A new function has been added; sUserID(). This returns the login name of the current user. This function and sMachineID are used by cogent scripts to record the username and machine ID in any standard data file so that the creator of the file can be identified if necessary.

Changes v1.17 to v1.19

1/ There are no changes between v1.17 and v1.19. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.16 to v1.17

1/ Bug fix - the time in microseconds as returned by sGetTime() was held as a 'long' and so wrapped round to a negative value after reaching 2*31 (2,147,483,648). This occurred after 2,147 seconds (31' 47"), an unacceptably short interval. This function now returns a floating point 'double' in units of seconds.

Changes v1.07 to v1.16

1/ There are no changes between v1.07 and v1.16. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.06 to v1.07

1/ A new data structure has been defined; CogStdData, which is returned by one of the new functions.

2/ A new function has been added; sGetData(). This returns the CogStdData structure. This function cannot be accessed from the matlab workspace but can be accessed from C code. It has been provided to interface with the GScnd function cgGetData which is the general interface for users to access low-level data.

3/ A new function has been added; sMachineID(). This returns the name and IP address of the computer that cogent is running on.

Changes v1.04 to v1.06

1/ There are no changes between v1.04 and v1.06. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.03 to v1.04

1/ A new global function has been added; CogStdVers(). This function cannot be called directly from Matlab. Its function is to provide a standard function (available from all the dlls) which can print out version information, usage information and a copyright message. This function also returns the version number as a numerical value. This function is called by the cgVers() function, callable from Matlab, which can be used to list the version numbers, usage descriptions and copyright messages of all the dlls.

2/ The mexGateway() function has been modified. It now recognises being called with the single text argument 'LoadLib' in which case it returns without any other action. This provides a way of calling each dll in a completely passive manner and is used to simply load the library into the matlab executable space. This is utilised in the GScnd matlab script cgLoadLib.m to load all cogent dlls ready for use. This is necessary because some dlls call each other and this only works infallibly if the called dll has already been loaded by Matlab.

Changes v1.02 to v1.03

1/ There are no changes between v1.02 and v1.03. The version number has simply been incremented to keep in step with GPrim.dll

Changes v1.01 to v1.02

1/ Function mexGateway() modified in subroutine PrintUsage() to allow 12 characters rather than 8 for function names.

Changes v1.00 to v1.01

1/ Most 'int' variables have been changed to 'long'. This is to avoid confusion between 16 bit and 32 bit 'int' variables. The size of a 'long' must always be 32 bits.

2/ Minor bug fix in mexGateway() function – trapping of too many arguments was being missed. Culprit was:-

```
else if ((nrhs < pComms[GatewayCom].rargsmin)||
   (nrhs < pComms[GatewayCom].rargsmax))

else if ((nlhs < pComms[GatewayCom].largsmin)||
   (nlhs < pComms[GatewayCom].largsmax))
```

Changed to:-

```
   (nrhs > pComms[GatewayCom].rargsmax))
   (nlhs > pComms[GatewayCom].largsmax))
```

3/ Added function sGetTime()

4/ Added functions sOnAbort() and sOutAbt().

5/ Minor changes involving capitalisation of some function names – e.g. sVers()

Table of contents

Introduction

General description

The CogStd.dll dynamically linked library (dll) is designed to provide a set of standardised functions to be called from other Matlab dlls. It provides basic functionality for the Stimulus presentation project "Cogent Graphics".

Matlab

The Matlab program package has been selected as a basic skeleton for Cogent Graphics because it provides a fully implemented programming and scripting language "ready-made". It also has the ability to incorporate user defined functions such as GPrim.dll. The design target for Cogent Graphics is that it should run under the cheapest commercially available release of Matlab which is currently the Student edition.

DevStudio and Visual C++

Microsoft's proprietary compiler, linker and integrated development environment (IDE), DevStudio, has been chosen for developing code because we have already had experience of it as a tool for medium to large project development and because it is available at a modest price to the educational sector. In particular the sophisticated remote debugging facility should prove very useful.

Compiling and Linking

Remote debugging with DevStudio

Remote debugging mode allows a host computer to display the debugger's windows while the other computer, the remote or target computer, displays output from the program being debugged. This is essential when debugging graphics code since the primary display is devoted to the executable rather than the debugger. There are four steps to remote debugging using DevStudio:-

1/ **Copy files to the remote computer**

Copy the following files to C:\Windows on the remote computer:-

MSVCMON.EXE	MSVCRT.DLL	TLN0T.DLL
MSVCP50.DLL	MSDIS100.DLL	DM.DLL

These files are in the DevStudio\SharedIDE\bin and Windows\System subfolders of the Visual C++ distribution (For Visual C++ 5.00)

In addition, Matlab should be installed on the remote computer.

Finally, a working directory should be created on the remote host which will contain all the linked dlls created on the host DevStudio process. Filesharing must be set up so that the host has full access to this working directory.

2/ **Configure the host computer**

The "Debug" tab of the Project Settings dialog box should be filled out as shown below. Here the "Executable for debug session" and "Remote executable path and file name" text boxes have been set up to run the Matlab executable on a remote host named "Mustard". Matlab has been installed on "Mustard" in the path "C:\matlab". The "Working directory" text box has been set to the path C:\matlab\toolbox\psychtb on "Mustard".

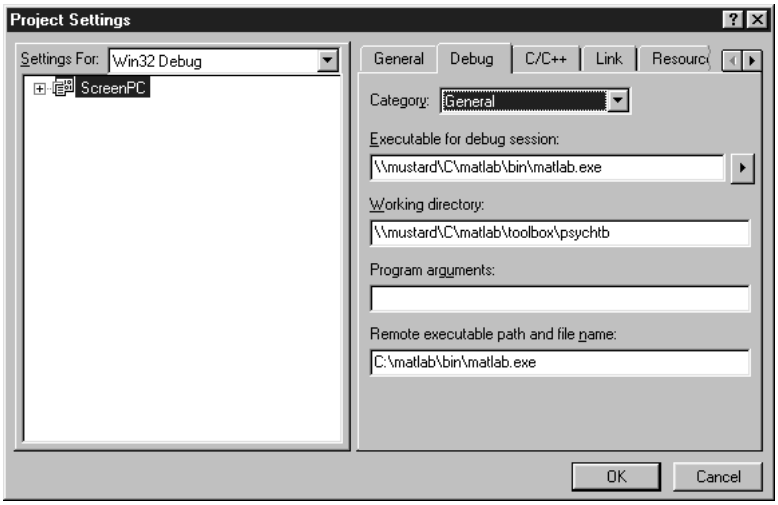

Fig B.1 Project settings Debug tab General Category

In addition, the "Link" tab of the Project Settings dialog box should be filled out as shown below. Here the "Output file name" text box has been set to use the same path that was specified in the "Working directory" textbox in the "Debug" tab as shown above.

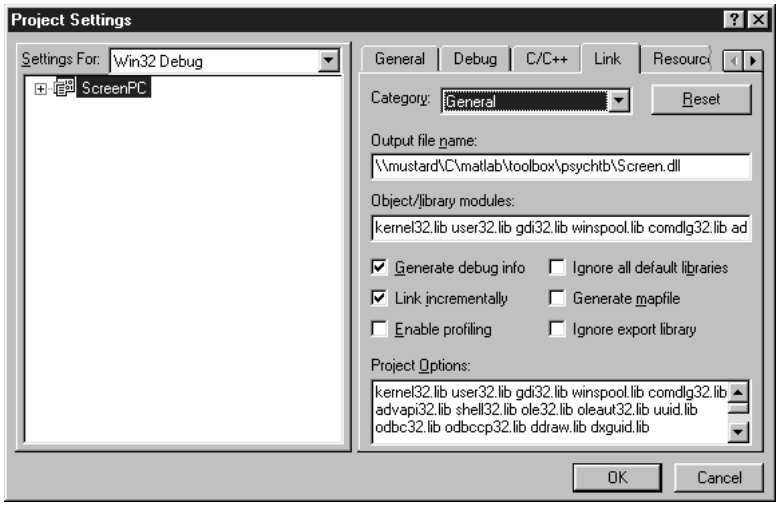

Fig B.2 Project settings Link tab General Category

Then select "Debugger Remote Connection" from the Build menu to display the "Remote Connection" dialog and set it as shown below:-

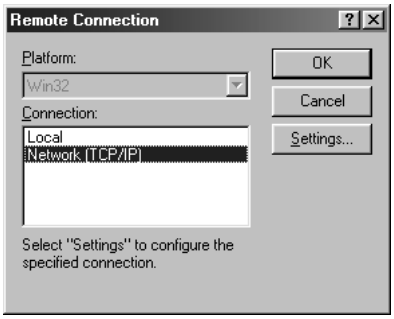

Fig B.3 Remote connection dialog box

3/ **Configure the remote computer**

Run the debugging monitor program on the remote computer by running MSVCMON.EXE from a DOS box:-

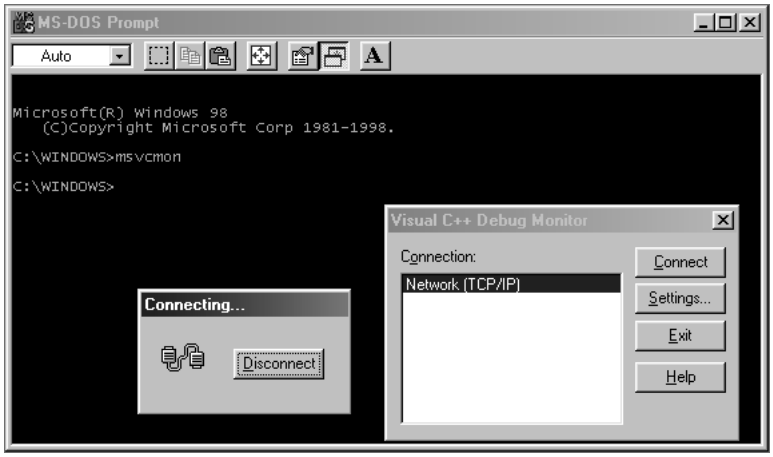

Fig B.4 Debugging monitor program

When ready to connect to the host, click the "Connect" button and the "Connecting..." dialog should appear to indicate that the remote machine is waiting to connect with a DevStudio debug session on a host.

4/ **Start the debugger**

After setting breakpoints, begin the debugger normally on the host machine. After a few seconds of chugging and whirring the debugging monitor dialogs on the remote machine should disappear and Matlab should start up in the specified working directory waiting for input. You may run a script at the Matlab command line, call a dll directly or put a "startup.m" Matlab script in the working directory which will be automatically executed when Matlab runs. For tidiness, the startup.m script can end with an "exit" statement to end the debug session if you require.

Compiling and linking to Matlab functions

In order to access mex functions such as mxGetPr() the dll must be linked with two files named mymeximports.lib and mymeximports.exp. These provide the information the linker needs to link your function calls with the mex function library.

The files mymeximports.lib and mymeximports.exp can be made using the following command from a DOS command window:-

lib /def:matlab\extern\include\matlab.def /out:mymeximports.lib

The Project Settings "Link" tab "Input" Category should have the "Object/Library modules" text box altered to include "mymeximports.lib" as shown in Fig B.5 above and you should add the Matlab include directory matlab\extern\include to the "Additional include directories" text box as shown in Fig B.6 above. The following Matlab include files should be included in your source code as required:-

```
#include "engine.h"
#include "fintrf.h"
#include "mat.h"
#include "matrix.h"
#include "mex.h"
#include "mwdebug.h"
#include "tmwtypes.h"
```

Matlab calling convention

A function to be called directly from Matlab can be implemented as a dynamically linked library (dll). The name of the dll is the name of the Matlab call. So for example, a Matlab function named "Screen" is implemented in a dll named "Screen.dll".

The entry point in the dll is of a standard form as shown below:-

```
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
}
```

The linker must be informed what functions are to be exported from the dll using a separate .DEF file containing the lines below:-

```
LIBRARY Screen.dll
EXPORTS mexFunction
```

The .DEF file must be included among the project source files.

There is an alternative way of doing this using the "__declspec(dllexport)" qualifier as a storage class modifier to mexFunction but then the Matlab include file mex.h must also be altered so that the mexFunction prototype also has this qualifier.

The arguments to `mexFunction` are defined as follows:-

```
const int mxMAXNAM = 32;

struct mxArray_tag
{
    char     name[mxMAXNAM];
    int      reserved1[2];
    void    *reserved2;
    int      number_of_dims;
    int      nelements_allocated;
    int      reserved3[3];
    union
    {
        struct
        {
            void  *pdata;
            void  *pimag_data;
            void  *reserved4;
            int    reserved5[3];
        }number_array;
    }data;
};

typedef struct mxArray_tag   mxArray;
typedef const  mxArray       CONSTmxArray;
```

The use of these arguments is described in the Matlab Application Program Interface Guide Chapter 2 "Creating C Language MEX-files".

CogStd

Library

Design

Abort, error, warning and text output

Output of text for abort, error, warning and information messages is passed to the CogStd library functions sOutAbt, sOutErr, sOutWrn and sOutStr respectively. Handling errors in this way has the following advantages:-

1/ A standard appearance and format for error messages.

2/ The possibility of saving all output text to a log file.

3/ The ability to separate Error and Warning messages. Warning messages are not usually displayed but they can be enabled in CogStd. This allows for a possible "Debugging mode" for output.

4/ There is a formalised way to abort the current script if necessary, cleaning up any loose ends in any dll along the way.

The policy for dealing with errors is that any errors encountered will be immediately output to the screen and as much useful ancilliary information as possible will be output in the text of the error message. The Abort mechanism is a way of indicating a fatal error in which the current script should be aborted and no further processing should take place.

Timing

The function sGetTime() provides the single timing reference for the program. It is placed in this library as a standard function to be used by all other dlls.

Version information

Since Cogent Graphics is built up from numerous sub-modules which are constantly being updated it is essential that each library must be able to identify itself.

The version function should be callable directly from Matlab and should identify the following items:-

Library name, Version number, Compilation date

The function `sVers()` typically produces the following output:-

CogStd v1.33 Compiled:Aug 12 2015

Typical sourcecode listing for the `sVers()` function is shown below. The `__DATE__` item is a VisualC preprocessor definition used here to automatically set the compilation date. The constants `LIB_NAME` and `LIB_VERS` are defined in an included header file in the sourcecode.

```
static const char  LIB_NAME[] = "CogStd";
static const long  LIB_VERS = 100;

void sVers(void)
{
    char  Buffer[256];

    sprintf(Buffer,"%s v%d.%02d Compiled:%s\n",
       LIB_NAME,LIB_VERS/100,LIB_VERS%100,__DATE__);

    sOutStr(Buffer);
}
```

There is also another version function named CogStdVers() which has no directly callable Matlab console interface. This function is provided for compatibility with the GScnd suite of functions, in particular the cgVers() function. The function is provided in a standard form for all dlls and can be called to print version, usage and copyright information.

```
long CogStdVers(long Flags)
{
    char  Buffer[256];

    if (Flags & VERF_VER)
       sVers();

    if (Flags & VERF_USG)
       mexGateway(LIB_NAME,CommandArray,
          sizeof(CommandArray)/sizeof(CommandArray[0]),
          0,NULL,0,NULL);

    if (Flags & VERF_CPY)
    {
       sprintf(Buffer,
          "%s v%d.%02d. Written by J.Romaya. Copyright © %s %s\n",
          LIB_NAME,LIB_VERS/100,LIB_VERS%100,__DATE__,CopyrightStr);
       sOutStr(Buffer);
    }
    return LIB_VERS;
}
```

Library contents

The CogStd.dll library contains the following functions:-

```
mexFunction   sVers       sOnAbort    sOutAbt     sEnable
sLogFil       sOutErr     sOutWrn     sOutStr     sGetTime
sOSErrStr     sMachineID  sGetData    sUserID     sPriority
sOSID
```

The function `mexFunction` can be called indirectly from the Matlab console command line by using the library name followed by a number of arguments in brackets:-

```
CogStd('sOutStr','This string to be output as text')
```

The general format of the call is as follows:-

```
CogStd('command',arg1,arg2,...argn)
```

Where `'command'` is one of the CogStd functions and arg1...argn are the associated arguments. All of the CogStd functions except `sOnAbort and sOSErrStr` may be called in this way.

Abort functions

The following functions are used to abort the matlab script and print an "abort" message:-

```
void sOnAbort(void (__cdecl *AbortFunction)(void));
void sOutAbt(char *Str);
```

The `sOnAbort()` function registers a cleanup function for each dll that will be called to clean up memory, display, filehandles etc… if the script has to be aborted. The cleanup function should take no arguments and return no value. The function uses the CMapPtrToPtr class to maintain a list of these cleanup functions:-

```
static CMapPtrToPtr     AbortFunctionMap;

void sOnAbort(void (__cdecl *AbortFunction)(void))
{
      AbortFunctionMap.SetAt(AbortFunction,AbortFunction);
}
```

The `sOutAbt()` function generates an "Abort message" from the supplied string and sends it to the current log file if there is one. It then calls each of the abort functions that have been registered with `sOnAbort()`. Finally it calls the Matlab function `mexErrMsgTxt()` to terminate the current script and print the abort message on the Matlab console window.

```
void sOutAbt(char *ErrMsg)
{
      void (__cdecl *AbortFunction)(void);
      POSITION    POS;
      char        AbortStr[256];

      sprintf(AbortStr,"\n*******\n*ABORT* %s\n*******\n",ErrMsg);

      if (pLogFile)
            fprintf(pLogFile,"%s",AbortStr);

      POS = AbortFunctionMap.GetStartPosition();

      while (POS)
      {
            AbortFunctionMap.GetNextAssoc(POS,
                  (void *&)AbortFunction,
                  (void *&)AbortFunction);
            AbortFunction();
      }
      mexErrMsgTxt(AbortStr);
}
```

Text output functions

The following functions are concerned with text output:-

```
void sEnable(const char *EnableString)
void sLogFil(const char *pLogFilName)
void sOutErr(const char *Lib,const char *Com,const char *Str);
void sOutWrn(const char *Lib,const char *Com,const char *Str);
void sOutStr(const char *Str);
```

The functions `sOutErr()` and `sOutWrn()` are designed to standardise the format for error and warning messages so that they appear as follows:-

```
ERR GPrim:ginit Invalid GLibName
```

Warning messages appear with `WRN` rather than `ERR`. In general the format of these messages is:-

```
ERR/WRN  LibraryName:Command ErrorMessage
```

Warning messages are printed out by default but they can be disabled by a call to `sEnable()`.

The `sEnable()` code checks its string argument for specific characters that turn certain attributes on. If the character is missing from the string the attribute is turned off. Currently the specific characters and attributes are:-

Character w/W Enables printing of warning messages.

```
void sEnable(const char *pEnableString)
{
      long        i,l;
      char  Buffer[256];

      OutputWarnings = false;

      l = strlen(pEnableString);

      for (i = 0;i < l;++i)
            switch (pEnableString[i])
            {
            case 'w':
            case 'W':
                  OutputWarnings = true;
                  break;

            default:
                  sprintf(Buffer,"Invalid flag %c",pEnableString[i]);
                  sOutErr(LIB_NAME,"sEnable",Buffer);
                  break;
            }
}
```

The functions `sOutErr()` and `sOutWrn()` actually call `sOutStr()` as shown below:-

```
void sOutErr(const char *Lib,const char *Com,const char *Str)
{
      char  Buffer[256];

      sprintf(Buffer,"ERR %s:%s %s\n",Lib,Com,Str);

      sOutStr(Buffer);
}

void sOutWrn(const char *Lib,const char *Com,const char *Str)
{
      char  Buffer[256];

      if (OutputWarnings)
      {
            sprintf(Buffer,"WRN %s:%s %s\n",Lib,Com,Str);

            sOutStr(Buffer);
      }
}
```

The `sOutStr()` function prints to the console but it also writes the same output to a file with the handle pLogFile if one has been defined using the `sLogFile()` function.

```
void sOutStr(const char *Str)
{
      printf(Str);

      if (pLogFile)
            fprintf(pLogFile,Str);
}
```

The `sLogFile()` function is responsible for a certain amount of housekeeping and bulletproofing. When first called it installs the function `Cleanup()` as the exit function for this library when Matlab exits It also registers the `Cleanup()` function as the Abort function using the `sOnAbort()` call. If a log file has already been opened, `sLogFile()` first closes it and then attempts to create a new one with the name specified in the pLogFilName argument. The first line in the log file identifies the file type and includes the date and time.

```
	static FILE *pLogFile;

	void sLogFil(const char *pLogFilName)
	{
		char		Buffer[256];
		time_t		t;

		if (!AtExitFlag)
		{
			sOnAbort(&Cleanup);
			atexit(&Cleanup);
			AtExitFlag = true;
		}

		if (pLogFile)
		{
			fclose(pLogFile);
			pLogFile = NULL;
		}

		if (pLogFilName[0])
		{
			pLogFile = fopen(pLogFilName,"at");

			if (!pLogFile)
			{
				sprintf(Buffer,"Unable to open log file:%s (%s)",
					pLogFilName,
					sOSErrStr(errno));
				sOutErr(LIB_NAME,"sLogFil",Buffer);
			}

		time(&t);
		fprintf(pLogFile,"Cogent Graphics log file
%s",asctime(localtime(&t)));
		}
	}
```

Timing functions

The function `sGetTime()` returns a time in seconds as a floating point double data item. If called with a non-negative argument it resets the second counter to that value. Typically the function should be called with a 'zero' argument at the beginning of the experiment to zero the timer. This should be combined with some kind of time-of-day timestamp to record the real world time. The function uses the Multimedia library timing system which is documented as being accurate down to a time resolution of 1mS. The timing resolution is set to 1mS using the `timeBeginPeriod()` function. This should be married with a corresponding `timeEndPeriod()` call and this is achieved by using the mmTimeResolution variable in conjunction with the `Cleanup()` function. The `Cleanup()` function is registered here with the `atexit()` and `sOnAbort()` function calls.

```
static long       mmTimeResolution = 0;
static long       mmTimeOffset;

double      sGetTime(double S)
{
      long  mS;

      if (!mmTimeResolution)
      {
            if (!AtExitFlag)
            {
                  sOnAbort(&Cleanup);
                  atexit(&Cleanup);
                  AtExitFlag = true;
            }

            mmTimeResolution = 1;
            timeBeginPeriod(mmTimeResolution);
            mmTimeOffset = -(long)(timeGetTime());
      }

      if (S >= 0)
      {
            mS = (long)(S*1000.);
            mmTimeOffset = mS - timeGetTime();
      }

      return ((double)(timeGetTime() + mmTimeOffset))/1000.;
                                    }
```

Error functions

The function `sOSErrStr()` returns a string describing the last operating system error encountered and is designed for use in constructing error messages. An example of its use is to be found in the following code fragment from `sLogFil()`:-

```
pLogFile = fopen(pLogFilName,"at");

if (!pLogFile)
{
      sprintf(Buffer,"Unable to open log file:%s (%s)",
            pLogFilName,
            sOSErrStr(errno));
      sOutErr(LIB_NAME,"sLogFil",Buffer);
}
```

The code constructs a string using the Visual C function strerror():-

```
char *sOSErrStr(long ErrNum)
{
      static char Buffer[256];

      sprintf(Buffer,"OSErr:%d %s",ErrNum,strerror(ErrNum));

      return Buffer;
}
```

Housekeeping

The function `Cleanup()` is responsible for cleaning up memory, filehandles, etc… . It is registered with the `atexit()` and `sOnAbort()` function calls in `sLogFile()` and `sGetTime()`.

```
static void Cleanup(void)
{
      if (pLogFile)
      {
            fclose(pLogFile);
            pLogFile = NULL;
      }

      if (mmTimeResolution)
      {
            timeEndPeriod(mmTimeResolution);
            mmTimeResolution = 0;
      }
}
```

The function `sPriority()` can be used to set the priority class for the current process. It is used to raise the priority of the cogent graphics process sufficiently so that the gFlip() command operates reliably.

```
long  sPriority(long NewClass)
{
      long  OldClass;
      HANDLE      hProcess;
      char  Buffer[256];

      hProcess = GetCurrentProcess();

      OldClass = GetPriorityClass(hProcess);

      switch (OldClass)
      {
      case NORMAL_PRIORITY_CLASS:
      case IDLE_PRIORITY_CLASS:
      case HIGH_PRIORITY_CLASS:
      case REALTIME_PRIORITY_CLASS:
            break;

      case 0:
            sprintf(Buffer,"GetPriorityClass() failed (%s)",
                  sOSErrStr(GetLastError()));
            sOutErr(LIB_NAME,"sPriority",Buffer);
            OldClass = 0;
            break;

      default:
            sprintf(Buffer,"Invalid GetPriorityClass() (%d)",OldClass);
            sOutErr(LIB_NAME,"sPriority",Buffer);
            OldClass = 0;
            break;
      }

      switch (NewClass)
      {
      case NORMAL_PRIORITY_CLASS:
      case IDLE_PRIORITY_CLASS:
      case HIGH_PRIORITY_CLASS:
      case REALTIME_PRIORITY_CLASS:
            if (!SetPriorityClass(hProcess,NewClass))
            {
                  sprintf(Buffer,"Unable to set priority class (%s)",
                        sOSErrStr(GetLastError()));
                  sOutErr(LIB_NAME,"sPriority",Buffer);
            }
            break;

      case 0:
            break;

      default:
            sOutErr(LIB_NAME,"sPriority","Invalid priority class");
            break;
      }

      return OldClass;
}
```

Interrogation

The function `sGetData()` cannot be called from the matlab workspace but only from C. It is designed to pass data from the CogStd library to the main Cogent Graphics data interrogation function `cgGetData`, which can return data from all the libraries. The function returns a pointer to a `CogStdData` structure:-

```
struct CogStdData
{
long  Version;
char  CogStdString[256];
};
```

This structure contains the following members:-

Version — The version number of the current release multiplied by 100. i.e. if the current version number is 1.33 this variable will contain the integer 133.

CogStdString — Version information for the library encoded as an ascii string.

The `sGetData()` function is shown below:-

```
void *sGetData(enum GetDatCod GDC,long Selector)
{
      char                          Buffer[256];
      static struct CogStdData      CSD;

      if (!DataInit)
      {
            sprintf(m_Data.CogStdString,"%s v%d.%02d Compiled:%s",
                  LIB_NAME,LIB_VERS/100,LIB_VERS%100,__DATE__);
            DataInit = true;
      }

      switch (GDC)
      {
      case GDC_CSD:
            CSD = m_Data;
            return &CSD;

      default:
            sprintf(Buffer,"Unknown data request %d",GDC);
            sOutErr(LIB_NAME,"sGetData",Buffer);
            return NULL;
      }
}
```

The function takes two arguments; `GDC` and `Selector`. The first argument is defined as below:-

```
enum GetDatCod
{
 GDC_CSD = 0,       // CogStdData
 GDC_GPD,           // GPrimData
 GDC_RAS,           // RAS structure
 GDC_DIB,           // DIB structure
 GDC_GSD,           // GScndData structure
 GDC_SPR,           // Sprite structure
 GDC_NUM
};
```

This allows various data items to be requested from the different libraries. The only type processed by this function is `GDC_CSD` (CogStdData request). The other `GDC` values request other data from other Cogent Graphics libraries and the `Selector` argument is sometimes used as well to select a particular data item.

Identification functions

Two functions have been provided so that a script can record the current user and identify the PC which is being used. These two pieces of information can be saved in any data file that is created so that the user may be identified at a later date.

The function `sMachineID()` returns an ascii string identifying the computer where Cogent Graphics is running. The string contains the machine name and, where possible, the IP address of the machine.

```
char *sMachineID(void)
{
 int         ret;
 WSADATA     wsaData;
 HOSTENT     *pHOSTENT;
 static WORD wVersionRequested = MAKEWORD(1,0);
 static char UD[] = "Undefined";
 static char Buffer[256];

 if (!gethostname(Buffer,sizeof(Buffer)))
       return &Buffer[0];

 switch (WSAGetLastError())
 {
 case WSANOTINITIALISED:
       if (!WSAStartup(wVersionRequested,&wsaData))
       {
             ret = gethostname(Buffer,sizeof(Buffer));
             if (!ret)
             {
                   pHOSTENT = gethostbyname(Buffer);
                   if (pHOSTENT)
                   {
                         if ((pHOSTENT->h_length ==
                               sizeof(struct in_addr))&&
                               (pHOSTENT->h_addr_list[0]))
                         {
                               strcat(Buffer," ");
                               strcat(Buffer,inet_ntoa
     (*((struct in_addr *)pHOSTENT->h_addr_list[0])));
                         }
                   }
             }
             WSACleanup();

             if (!ret)
                   return &Buffer[0];
       }
       break;
 }

 return &UD[0];
}
```

The function `sUserID()` returns an ascii string identifying the user login that was used for the current session.

```
char *sUserID(void)
{
 static char       UD[] = "Undefined";
 static char       Buffer[256];
 unsigned long     l;

 l = sizeof(Buffer);

 if (GetUserName(Buffer,&l))
       return &Buffer[0];

 return &UD[0];
}
```

The function sDXVer() returns the version number of the currently installed DirectX package, multiplied by 100.

```
 long sDXVer()
 {
       unsigned long     Version;
       char  Buffer[256];

       static long MasterVersion = -1;

       extern HRESULT GetDXVersion(DWORD* pdwDirectXVersion,
             TCHAR* strDirectXVersion, int cchDirectXVersion );

       if (MasterVersion == -1)
       {
             if (GetDXVersion(&Version,Buffer,sizeof(Buffer)) != S_OK)
                   return 0;

             MasterVersion = (long)(HIWORD(Version)*100 + LOWORD(Version));
       }

       return MasterVersion;
 }
```

The function GetDXVersion has been copied from the DirectX SDK sample code.

The function sOSID() returns the name of the currently installed operating system as an ascii string. This complicated function has been copied from the Windows Software Developers Kit (Windows SDK).

Cogent Graphics Matlab console interface

The `mexFunction()` in this library uses a call to the function `mexGateway()`.

This has a number of advantages:-

i/ Standardised calling of functions from the Matlab console.
ii/ Standardised error checking and parsing of arguments.
iii/ Built-in listing of all functions available from the Matlab console.

There are a number of steps to follow to use the function:-

1/ Define the functions to be made available from the Matlab console and create an array of Command structures, one structure for each function:-

```
struct Command      // Defined in CogStd.h
{
     char *pName;
     char *pArgDesc;
     long rargsmin;
     long rargsmax;
     long largsmin;
     long largsmax;
     void (*MexFnc)(int nlhs, mxArray *plhs[], int nrhs,
                    const mxArray *prhs[]);
};

const struct Command CommandArray[] =
{
     {"sEnable",    "arg1=EnableStr",     1,1,    0,0,   &mex_sEnable},
     {"sLogFil",    "arg1=LogFile name",  1,1,    0,0,   &mex_sLogFil},
     {"sOSErrStr", "no arguments",        0,0,    1,1,   &mex_sOSErrStr},
     {"sOutStr",    "arg1=Str",           1,1,    0,0,   &mex_sOutStr},
     {"sOutWrn",    "arg1=Lib,arg2=Com",  3,3,    0,0,   &mex_sOutWrn},
     {"sOutErr",    "arg1=Lib,arg2=Com",  3,3,    0,0,   &mex_sOutErr},
     {"svers", "no arguments",    0,0,    0,0,    &mex_svers}
};
```

The Command structure contains a set of variables to define the function to be made available from the Matlab console:-

pName The name of the command

pArgDesc A description of the arguments it takes

rargsmin/ rargsmax The range (min and max) of numbers of arguments that this function takes.

largsmin/ largsmax The range (min and max) of numbers of return values that this function produces.

MexFnc A mex gateway interface function that you must provide for further argument parsing and generation of return values.

2/ Define a mex gateway interface function

Each command callable from the mex interface must have an interface function as in the example below:-

```
static void mex_sLogFil(int nlhs,mxArray *plhs[],int nrhs,
                        const mxArray *prhs[])
{
     long      Status;
     char LogFileName[256];

     if (!mxIsChar(prhs[0]))
          mexError("arg1 must be text");    // REPORT AN ERROR
     else
     {
          Status = mxGetString(prhs[0],LogFileName,
                               sizeof(LogFileName));

          if (Status != 0)
               mexError("Unable to convert arg1 to text");
          else
               sLogFil(LogFileName);        // CALL UNDERLYING FUNCTION
     }
```

This function is responsible for checking that the supplied arguments are of a valid type and reporting any errors of that sort and then for calling the underlying function. The function is also responsible for creating any return values that are to be passed back to the Matlab console.

The call to the function `mexError()` shown above can only be used from a mex interface function. It relies on the function `mexGateway()` having been called and having correctly initialised the variables `LibraryName` and `CommandName` to allow a standard error message to be generated showing the correct library name and command name:-

```
void mexError(const char *Str)
{
     sOutErr(LibraryName,CommandName,Str);
}
```

3/ Define your `mexFunction()` to call `mexGateway()` as shown below:-

```
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,
       const mxArray *prhs[])
{
     mexGateway(LIB_NAME,
        CommandArray,
        sizeof(CommandArray)/sizeof(CommandArray[0]),
        nlhs,plhs,nrhs,prhs);
}
```

`LIB_NAME` is the name of your calling library and you then pass the `CommandArray` and the number of elements followed by the four standard mex arguments.

The `mexGateway()` function performs the following tasks:-

i/ It copies the library name to the variable `LibraryName` for use by the `mexError()` function.

ii/ If there are no arguments it prints the usage guide for this library based on the information in the `Command` array. This lists the functions available on the Matlab console as well as the argument descriptions.

iii/ The first argument should be a string which matches one of the command names. This command is identified or otherwise an error message is printed. The correctly identified command name is copied to the variable `CommandName` for use by the `mexError()` function. If the first argument is not one of the command names but is 'LoadLib' instead, the function simply returns. This provides a mechanism for simply loading the library into the matlab executable space without producing any output.

iv/ The number of arguments and return values is checked against the information in the relevant `Command` structure. If there is a discrepancy an error message is printed. Otherwise the corresponding mex interface function is called and passed the standard Matlab arguments which have been altered to remove the first right hand argument (the library command string). It is up to the interface function to check the types and values of the arguments it is passed.

The rather lengthy source listing is shown below:-

```
void mexGateway(const char *pLibName,const struct Command *pComms,
      long nComms,int nlhs,mxArray *plhs[],int nrhs,
      const mxArray *prhs[])
{
      long  i,status,GatewayCom;
      char  Buffer[256];

      strcpy(LibraryName,pLibName);
/*
 * If there are no arguments just print the usage guide
 */
      if (nrhs < 1)
      {
            PrintUsage(pComms,nComms);
            return;
      }
/*
 * First argument must be text
 */
      else if (!mxIsChar(prhs[0]))
      {
            SyntaxError("Command must be text");
            PrintUsage(pComms,nComms);
      }
      else
      {
            status = mxGetString(prhs[0],CommandName,
                  sizeof(CommandName));

            nrhs--;
            prhs++;
```

```
                if (status != 0)
                      SyntaxError("Unable to convert command to text");
                else
                {
                      GatewayCom = -1;
                      for (i = 0;i < nComms;++i)
                            if (!stricmp(pComms[i].pName,CommandName))
                            {
                                  GatewayCom = (enum GPrimCommand)i;
                                  break;
                            }

                      if (GatewayCom == -1)
                      {
                            if (!nrhs)
                                  if (!stricmp(CommandName,"LoadLib"))
                                        return;

                            sprintf(Buffer,"Invalid command %s",
                                  CommandName);
                            SyntaxError(Buffer);
                      }

                      else if ((nrhs < pComms[GatewayCom].rargsmin)||
                            (nrhs > pComms[GatewayCom].rargsmax))
                      {
                            if (pComms[GatewayCom].rargsmin ==
                                  pComms[GatewayCom].rargsmax)
                                  sprintf(Buffer,
                                        "expects %d args, not %d",
                                        pComms[GatewayCom].rargsmin,
                                        nrhs);
                            else
                                  sprintf(Buffer,
                                        "expects %d to %d args, not %d",
                                        pComms[GatewayCom].rargsmin,
                                        pComms[GatewayCom].rargsmax,nrhs);

                            mexError(Buffer);
                      }

                      else if ((nlhs < pComms[GatewayCom].largsmin)||
                            (nlhs > pComms[GatewayCom].largsmax))
                      {
                            if (pComms[GatewayCom].largsmin ==
                                  pComms[GatewayCom].largsmax)
                                  sprintf(Buffer,
                                        "produces %d args, not %d",
                                        pComms[GatewayCom].largsmin,nlhs);
                            else
                                  sprintf(Buffer,
                                        "produces %d to %d args, not %d",
                                        pComms[GatewayCom].largsmin,
                                        pComms[GatewayCom].largsmax,nlhs);

                            mexError(Buffer);
                      }

                      else
                            pComms[GatewayCom].MexFnc(nlhs,plhs,nrhs,prhs);
                }
          }
}
```

Separation of Matlab-specific code

I have tried to place all matlab-specific code in separate source modules in order to simplify any future implementations of the program which are not based on Matlab. Such source modules usually have a "mex_" prefix to their name. These modules contain the standard mex gateway function interface code for each dll. The CogStd library however also has another source module with a "mex2_" prefix. This contains additional mex-specific code including the mexGateway() and mexError() functions which provide the standardised interface.

Function Specifications

long CogStdVers(long Flags)

This function cannot be called directly from the matlab console.

Arguments

Flags — This argument selects what information is to be printed out to the console. The Flags argument can be the numerical sum of the following components:-

1 = Print version information
2 = Print usage information
4 = Print copyright information

Return Values

This function returns the version number of the CogStd library as a numerical value. Version 1.33 will be returned as the integer 133.

Remarks

This function is included as a standard function callable from all dlls. It provides a mechanism for the software to check that all dlls have the same version number and are therefore mutually compatible. It is principally used by the function GScnd:cgVers().

void mexError(char *Str)

Arguments

char *Str — An error message to print out on the Matlab console. If this function has been used correctly the call will have been made from a mex interface function through a call to `mexGateway()` (which makes an internal note of the calling library and command). The output error message automatically displays the calling library and command.

Return Values

This function does not return any values.

Remarks

This function cannot be called from the Matlab console. It should only be called from a mex interface function as described in Section C above under the heading "Cogent Graphics Matlab console interface".

Errors

This function does not generate any error messages.

void mexGateway(const char *pLibName,const struct Command *pComms, long nComms,int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])

Arguments
The arguments to this function require a detailed explanation and are discussed in Section C above under the heading "Cogent Graphics Matlab console interface".

Return Values
This function does not return any values.

Remarks
This function cannot be called from the Matlab console. It should only be called from the mexFunction() of your dll as described in Section C above under the heading "Cogent Graphics Matlab console interface".

Errors
This function generates various error messages connected with processing and parsing Matlab console arguments.

long sDXVer() xv=CogStd('sDXVer')

Arguments
This function takes no arguments.

Return Values
This function returns the version number of the currently installed DirectX package multiplied by 100.

Errors
This function will generate an error message if called from Matlab with incorrect arguments.

sEnable(char *EnableString) CogStd('sEnable','EnableString')

Arguments

EnableString	A string of specific characters, the presence or absence of which turn certain attributes on or off respectively. Possible values are:- w/W - Enable printing of warning messages.

Return Values
This function returns no values.

Remarks
This function can be used to enable or disable certain attributes according to the EnableString argument as detailed above.

Errors
This function will generate an error message if called from Matlab with incorrect arguments or if an unrecognised value for EnableString is used.

void *sGetData(enum GetDatCod GDC, long Selector)

This function cannot be called directly from the matlab console.

Arguments

GDC — This code defines what data is being requested. The possible values are defined in CogStd.h. This function only accepts the code GDC_CSD.

Selector — This variable is ignored by this function. It is included here for future expansion and for compatibility with the GetData functions of other Cogent Graphics libraries.

Return Values

This function returns a pointer to a CogStdData structure. This structure is defined in CogStd.h.

```
struct CogStdData
{
 long  Version;
 char  CogStdString[256];
};
```

The Version parameter contains the version number of the CogStd library multiplied by 100. So, for example, version 1.33 would have the number 133 here. The CogStdString variable contains the library identification in the form of an ascii string.

Remarks

This function cannot be called directly from Matlab but is designed to be called from the cgGetData() function in the GScnd suite of functions. That function can be used to access all low-level data from the constituent Cogent Graphics libraries.

Errors

This function will generate an error message if called from Matlab with incorrect arguments

double sGetTime(double S) S = CogStd('sGetTime',S)

Arguments

S — If this value is negative it is ignored. If it is positive or zero the second counter is reset to this value. When this function is called for the first time the second counter is reset to zero.

Return Values

This function returns the current value of the second counter.

Remarks

Currently the precision of the second timer is 0.001 seconds.

Errors

This function will generate an error message if called from Matlab with incorrect arguments

void sLogFil(char *LogFilName) CogStd('sLogFil','LogFilName')

Arguments

LogFilName	The name of a log file to create which will then have a copy of all text written to the Matlab console. An empty string switches logging off.

Return Values
This function returns no values.

Remarks
This function is used to create a log file which will then have a copy of all text written to the Matlab console. Logging may be switched off again by calling this function with an empty string for the LogFilName argument.

Errors
This function will generate an error message if called from Matlab with incorrect arguments or if an error is encountered when trying to create the log file.

char *sMachineID(void) CogStd('sMachineID')

Arguments
This function takes no arguments

Return Values
This function returns an ascii string identifying the computer on which Cogent Graphics is running. The machine name is given together with the machine IP address if available.

Remarks
This function is used to identify the current computer.

Errors
This function will generate an error message if called from Matlab with incorrect arguments.

void sOnAbort (void (__cdecl *AbortFunction)(void))

Arguments

AbortFunction	This argument is the address of a cleanup function for each dll. The cleanup function should release any allocated memory, close any open filehandles and perform other similar tidying up functions for the dll.

Return Values
This function returns no values.

Remarks
Call this function to register a cleanup function for your dll. This cleanup function will be called if the sOutAbt() function is ever called to terminate a Matlab script.

Errors
This function does not generate an error message.

char *sOSErrStr(long ErrNum) a = CogStd('sOSErrStr',ErrNum)

Arguments

ErrNum	The OS error number to describe.

Return Values
This function returns a string describing the last OS error encountered.

Remarks
This function is further described in Section C above under the heading "Error functions".

Errors
An error message is generated if the call from Matlab had incorrect arguments.

char *sOSID(void) s = CogStd('sOSID')

Arguments
This function takes no arguments.

Return Values
This function returns a string naming the currently installed operating system.

Remarks
This function is used to identify the currently installed operating system.

Errors
This function will generate an error message if called from Matlab with incorrect arguments.

void sOutAbt(char *Str) CogStd('sOutAbt','Str')

Arguments
Str The abort message to print out

Return Values
This function does not return. It aborts the current Matlab script immediately, printing out the error message on the Matlab console window.

Remarks
This function is used to terminate execution of a Matlab script due to a fatal error. When invoked it writes the abort message to the logfile if present and then calls each of the abort functions that have been registered using sOnAbort(). Finally it terminates execution of the current Matlab script, printing out the abort message onto the Matlab console window.

Errors
An error message is generated if the call from Matlab had incorrect arguments.

void sOutErr(char *Lib,char *Com,char *Str) CogStd('sOutErr','Lib','Com','Str')

Arguments
Lib The name of the library making this call.
Com The name of the library command making this call.
Str The error message to print out.

Return Values
This function returns no values.

Remarks
This function prints a standard error message out on the Matlab console. If a log file has been specified using `sLogFil()` the message will be output there as well.

A typical error message appears as shown below:-

```
ERR GPrim:ginit Invalid GLibName

(ERR LibraryName:Command ErrorMessage)
```

Errors
An error message is generated if the call from Matlab had incorrect arguments.

void sOutStr(char *Str) CogStd('sOutStr')

Arguments
Str The error message to print out.

Return Values
This function returns no values.

Remarks
This function prints a message out on the Matlab console. If a log file has been specified using `sLogFil()` the message will be output there as well.

Errors
An error message is generated if the call from Matlab had incorrect arguments.

void sOutWrn(char *Lib,char *Com,char *Str)

CogStd('sOutWrn','Lib','Com','Str')

Arguments

Lib	The name of the library making this call.
Com	The name of the library command making this call.
Str	The warning message to print out.

Return Values

This function returns no values.

Remarks

This function prints a standard warning message out on the Matlab console. Warning messages are displayed by default but they may be disabled by an explicit call to `sEnable()`. If a log file has been specified using `sLogFil()` the message will be output there as well.

A typical warning message appears as shown below:-

```
WRN GPrim:ginit Incorrect grommet diameter

(WRN LibraryName:Command WarningMessage)
```

Errors

An error message is generated if the call from Matlab had incorrect arguments.

long sPriority(long NewClass) CogStd('sPriority',NewClass)

Arguments

NewClass — When called from Matlab the NewClass argument should be one of the following strings:- "idle", "normal", "high" or "realtime". When called at the program level it should be a long integer set to one of the following values:-

IDLE_PRIORITY_CLASS	NORMAL_PRIORITY_CLASS
HIGH_PRIORITY_CLASS	REALTIME_PRIORITY_CLASS

Return Values

This function return the previous priority class as a string when called from Matlab or as a long integer when called at the program level.

Remarks

Use this function to set the priority class of the current process. It is necessary to set the priority class to "high" when using cogent graphics under Windows2000 to improve the reliability of the gFlip() command in GPrim. This is normally done by the start_cogent command. This alternative is provided here so that the current cogent graphics distribution is a self contained entity. Take care when using the "realtime" mode as it may adversely affect system performance.

Errors

An error message is generated if the call from Matlab had incorrect arguments or if there was an error encountered while executing the function.

char *sUserID(void) CogStd('sUserID')

Arguments
This function takes no arguments

Return Values
This function returns an ascii string identifying the user login name.

Remarks
This function is used to identify the current user and this information may be saved in any data file created by a script to identify who created it.

Errors
This function will generate an error message if called from Matlab with incorrect arguments.

void sVers(void) CogStd('sVers')

Arguments
This function takes no arguments.

Return Values
This function returns no values.

Remarks
This function generates a message identifying the current version of this library.

The function typically produces the following output:-

CogStd v1.33 Compiled:Aug 12 2015

Errors
An error message is generated if the call from Matlab had incorrect arguments.

Distribution

Media

Web distribution

The Cogent Graphics website is to be found at the following URL:-

http://www.vislab.ucl.ac.uk/cogent.php