Introduction

About our APIs

The core of Ghostscript is written in `C`, but also supports language bindings for the following programming languages:

- `C#`
- `Java`
- `Python`

All of the above languages have equivalent methods as defined in the C API. `Java` and `C#` provide additional helper methods to make the use of the API easier for certain applications. These languages also provide example viewers that make use of these methods.

This developer documentation is organized by programming language type and includes API reference and sample code.

The C API

Ghostscript has been in development for over thirty years and is written in `C`. The API has evolved over time and is continually being developed. The language bindings into Ghostscript will attempt to mirror this evolution and match the current C API as much as possible.

Licensing

Before using Ghostscript, please make sure that you have a valid license to do so. There are two available licenses; make sure you pick the one whose terms you can comply with.

Open Source license

If your software is open source, you may use Ghostscript under the terms of the GNU Affero General Public License.

This means that all of the source code for your complete app must be released under a compatible open source license!

It also means that you may not use any proprietary closed source libraries or components in your app.

Please read the full text of the AGPL license agreement from the FSF web site

If you cannot or do not want to comply with these restrictions, you must acquire a commercial license instead.

FIND OUT MORE

Commercial license

If your project does not meet the requirements of the AGPL, please contact our sales team to discuss a commercial license. Each Artifex commercial license is crafted based on your individual use case.

CONTACT US

Building Ghostscript

In order to use Ghostscript language bindings firstly Ghostscript must be built as a shared library for your platform.

The following built libraries are required for these respective platforms:

Platform	Ghostscript library files
Windows 32-bit	`gpdldll32.dll` `gsdll32.dll`
Windows 64-bit	`gpdldll64.dll` `gsdll64.dll`
MacOS	`libgpdl.dylib` `libgs.dylib`
Linux / OpenBSD	`libgpdl.so` `libgs.so`

> **NOTE**
>
> The actual filenames on MacOS will be appended with the version of Ghostscript with associated symlinks.

Building on Windows

To build the required DLLs, load `/windows/ghostpdl.sln` into Visual Studio, and select the required architecture from the drop down - then right click on 'ghostpdl' in the solution explorer and choose "Build".

Building on MacOS or Linux / OpenBSD

Firstly run the `autogen.sh` script from the command line to create the required configuration files followed by `make so` to build the shared libraries. The scripts also depend on having both `autoconf` and `automake` installed on your system. [1]

autoconf & automake

If this software is not already on your system (usually this can be found in the following location: `usr/local/bin` , but it could be located elsewhere depending on your setup) then it can be installed from your OS's package system.

Alternatively, it can be installed from GNU here:

https://www.gnu.org/software/autoconf/

https://www.gnu.org/software/automake/

Or, it can be installed via Brew by running:

```
brew install autoconf automake
```

Once built, these libraries can be found in your `ghostpdl/sobin/` or `ghostpdl/sodebugbin` location depending on your build command.

> **NOTE**
>
> For full detailed instructions on how to build your Ghostscript library see here.

Demo code

About

Please locate the `demos` folder in your `ghostpdl` source code download from the GhostPDL repository to find sample code demonstrating the language bindings in action.

C# overview

About

In the GhostPDL repository a sample `C#` project can be found in `/demos/csharp`.

Within this project the following namespaces and corresponding `C#` files are of relevance:

- GhostAPI `ghostapi.cs`
- GhostNET `ghostnet.cs`
- GhostMono `ghostmono.cs`

Platform & setup

Building Ghostscript

Ghostscript should be built as a shared library for your platform.

See Building Ghostscript.

GhostAPI

`GhostAPI` is the main wrapper responsible for bridging over to the `C` library and ensuring that the correct DLLs are imported.

`GhostAPI` contains the `ghostapi` class which *does not* need to be instantiated as it provides `public static` methods. These methods, which mirror their `C` counterparts, are as follows:

Method	Description
gsapi_revision	Returns the revision numbers and strings of the Ghostscript interpreter library
gsapi_new_instance	Create a new instance of Ghostscript
gsapi_delete_instance	Destroy an instance of Ghostscript
gsapi_set_stdio_with_handle	Set the callback functions for `stdio`, together with the handle to use in the callback functions
gsapi_set_stdio	Set the callback functions for `stdio`
gsapi_set_poll_with_handle	Set the callback function for polling, together with the handle to pass to the callback function
gsapi_set_poll	Set the callback function for polling
gsapi_set_display_callback	*deprecated*
gsapi_register_callout	This call registers a callout handler
gsapi_deregister_callout	This call deregisters a previously registered callout handler
gsapi_set_arg_encoding	Set the encoding used for the interpretation of all subsequent args supplied via the gsapi interface on this instance
gsapi_set_default_device_list	Set the string containing the list of default device names

Method	Description
gsapi_get_default_device_list	Returns a pointer to the current default device string
gsapi_init_with_args	Initialise the interpreter
gsapi_run_*	Wildcard for various "run" methods
gsapi_exit	Exit the interpreter
gsapi_set_param	Set a parameter
gsapi_get_param	Get a parameter
gsapi_enumerate_params	Enumerate the current parameters
gsapi_add_control_path	Add a (case sensitive) path to one of the lists of permitted paths for file access
gsapi_remove_control_path	Remove a (case sensitive) path from one of the lists of permitted paths for file access
gsapi_purge_control_paths	Clear all the paths from one of the lists of permitted paths for file access
gsapi_activate_path_control	Enable/Disable path control
gsapi_is_path_control_active	Query whether path control is activated or not

GhostNET

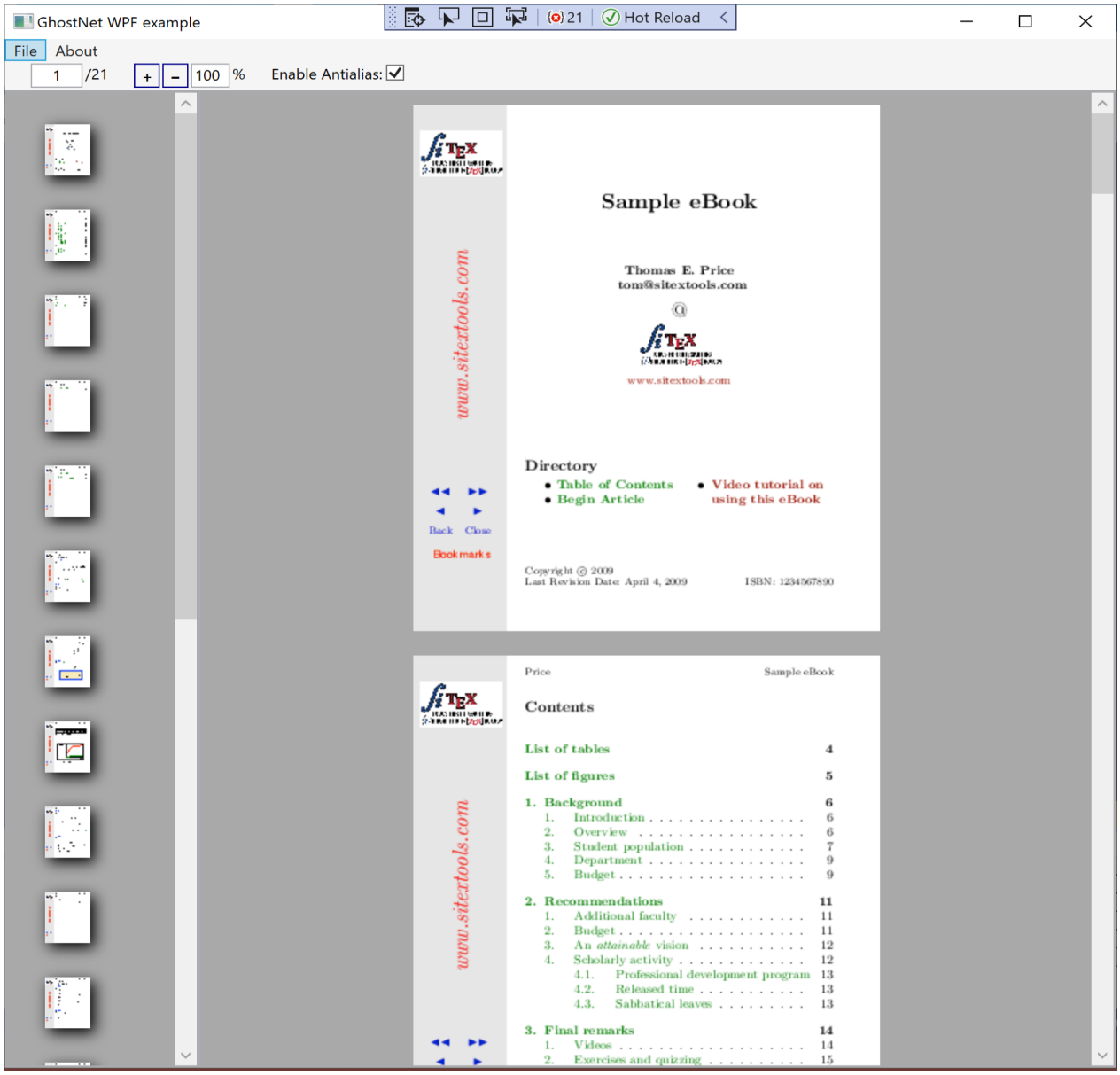
`GhostNET` is the .NET interface into `GhostAPI`. It exemplifies how to do more complex operations involving multiple API calls and sequences.

GhostNET WPF example

In `demos/csharp/windows/ghostnet.sln` there is a sample C# demo project.

This project can be opened in Visual Studio and used to test the Ghostscript API alongside a UI which handles opening PostScript and PDF files. The sample application here allows for file browsing and Ghostscript file viewing.

Below is a screenshot of the sample application with a PDF open:



GhostMono

`GhostMono` is the Mono equivalent of `GhostNET` and as such has no dependancy on a Windows environment.

GhostAPI

About

`GhostAPI` is the `C#` bridge into the Ghostscript `C` library.

`GhostAPI` contains some essential structs and enums as well as a static class for some constants, finally it contains the main GSAPI class which holds the key methods which interface with the `C` library.

Structs and Enums

gsapi_revision_t

This `struct` is used to contain information pertinent to the version of Ghostscript.

```
public struct gsapi_revision_t
{
    public IntPtr product;
    public IntPtr copyright;
    public int revision;
    public int revisiondate;
}
```

gs_set_param_type

```
public enum gs_set_param_type
{
    gs_spt_invalid = -1,
    gs_spt_null =   0, /* void * is NULL */
    gs_spt_bool =   1, /* void * is NULL (false) or non-NULL (true) */
    gs_spt_int = 2, /* void * is a pointer to an int */
    gs_spt_float = 3, /* void * is a float * */
    gs_spt_name = 4, /* void * is a char * */
    gs_spt_string = 5, /* void * is a char * */
    gs_spt_long =   6, /* void * is a long * */
    gs_spt_i64 = 7, /* void * is an int64_t * */
    gs_spt_size_t = 8, /* void * is a size_t * */
    gs_spt_parsed = 9, /* void * is a pointer to a char * to be parsed */
    gs_spt_more_to_come = 1 << 31
};
```

gsEncoding

```
public enum gsEncoding
{
    GS_ARG_ENCODING_LOCAL = 0,
    GS_ARG_ENCODING_UTF8 = 1,
    GS_ARG_ENCODING_UTF16LE = 2
};
```

Constants

Constants are stored in the static class `gsConstants` for direct referencing.

gsConstants

```
static class gsConstants
{
    public const int E_QUIT = -101;
    public const int GS_READ_BUFFER = 32768;
    public const int DISPLAY_UNUSED_LAST = (1 << 7);
    public const int DISPLAY_COLORS_RGB = (1 << 2);
    public const int DISPLAY_DEPTH_8 = (1 << 11);
    public const int DISPLAY_LITTLEENDIAN = (1 << 16);
    public const int DISPLAY_BIGENDIAN = (0 << 16);
}
```

GSAPI

Methods contained within are explained below.

`gsapi_run_*` and `gsapi_exit` methods return an `int` code which can be interpreted as follows:

code	status
`0`	no error
`gsConstants.E_QUIT`	"quit" has been executed. This is not an error. gsapi_exit must be called next
`<0`	error

> **NOTE**
>
> For full detail on these return code please see:
> https://www.ghostscript.com/doc/current/API.htm#return_codes

> **NOTE**
>
> All GSAPI methods aside from gsapi_revision and gsapi_new_instance should pass an instance of Ghostscript as their first parameter with `IntPtr instance`

gsapi_revision

This method returns the revision numbers and strings of the Ghostscript interpreter library; you should call it before any other interpreter library functions to make sure that the correct version of the Ghostscript interpreter has been loaded.

```
public static extern int gsapi_revision(ref gsapi_revision_t vers,
                                                    int size);
```

> **NOTE**
>
> The method should write to a reference variable which conforms to the struct gsapi_revision_t.

gsapi_new_instance

Creates a new instance of Ghostscript. This instance is passed to most other GSAPI methods. Unless Ghostscript has been compiled with the `GS_THREADSAFE` define, only one instance at a time is supported.

```
public static extern int gsapi_new_instance(out IntPtr pinstance,
                                            IntPtr caller_handle);
```

> **NOTE**
>
> The method returns a pointer which represents your instance of Ghostscript.

gsapi_delete_instance

Destroy an instance of Ghostscript. Before you call this, Ghostscript must have finished. If Ghostscript has been initialised, you must call gsapi_exit beforehand.

```
public static extern void gsapi_delete_instance(IntPtr instance);
```

```
GSAPI.gsapi_delete_instance(gsInstance);
gsInstance = IntPtr.Zero;
```

gsapi_set_stdio_with_handle

Set the callback functions for `stdio`, together with the handle to use in the callback functions. The `stdin` callback function should return the number of characters read, 0 for EOF, or -1 for error. The `stdout` and `stderr` callback functions should return the number of characters written.

> **NOTE**
>
> These callbacks do not affect output device I/O when using "%stdout" as the output file. In that case, device output will still be directed to the process "stdout" file descriptor, not to the `stdio` callback.

```
public static extern int gsapi_set_stdio_with_handle(IntPtr instance,
                                      gs_stdio_handler stdin,
                                      gs_stdio_handler stdout,
                                      gs_stdio_handler stderr,
                                                IntPtr caller_handle);
```

gsapi_set_stdio

Set the callback functions for `stdio`. The handle used in the callbacks will be taken from the value passed to gsapi_new_instance. Otherwise the behaviour of this function matches gsapi_set_stdio_with_handle.

```
public static extern int gsapi_set_stdio_with_handle(IntPtr instance,
                                      gs_stdio_handler stdin,
                                      gs_stdio_handler stdout,
                                      gs_stdio_handler stderr);
```

gsapi_set_poll_with_handle

Set the callback function for polling, together with the handle to pass to the callback function. This function will only be called if the Ghostscript interpreter was compiled with `CHECK_INTERRUPTS` as described in `gpcheck.h`.

The polling function should return zero if all is well, and return negative if it wants ghostscript to abort. This is often used for checking for a user cancel. This can also be used for handling window events or cooperative multitasking.

The polling function is called very frequently during interpretation and rendering so it must be fast. If the function is slow, then using a counter to `return 0` immediately some number of times can be used to reduce the performance impact.

```
public static extern int gsapi_set_poll_with_handle(IntPtr instance,
                                                    gsPollHandler pollfn,
                                                           IntPtr caller_handle);
```

```

```

gsapi_set_poll

Set the callback function for polling. The handle passed to the callback function will be taken from the handle passed to gsapi_new_instance. Otherwise the behaviour of this function matches gsapi_set_poll_with_handle.

```
public static extern int gsapi_set_poll(IntPtr instance,
                                        gsPollHandler pollfn);
```

gsapi_set_display_callback

This call is deprecated; please use gsapi_register_callout to register a callout handler for the display device in preference.

```
public static extern int gsapi_set_display_callback(IntPtr pinstance,
                                                    IntPtr caller_handle);
```

gsapi_register_callout

This call registers a callout handler.

```
public static extern int gsapi_register_callout(IntPtr instance,
                                                gsCallOut callout,
                                                   IntPtr callout_handle);
```

gsapi_deregister_callout

This call deregisters a callout handler previously registered with gsapi_register_callout. All three arguments must match exactly for the callout handler to be deregistered.

```
public static extern int gsapi_deregister_callout(IntPtr instance,
                                                  gsCallOut callout,
                                                     IntPtr callout_handle);
```

gsapi_set_arg_encoding

Set the encoding used for the interpretation of all subsequent arguments supplied via the `GhostAPI` interface on this instance. By default we expect args to be in encoding `0` (the 'local' encoding for this OS). On Windows this means "the currently

selected codepage". On Linux this typically means `utf8`. This means that omitting to call this function will leave Ghostscript running exactly as it always has. Please note that use of the 'local' encoding is now deprecated and should be avoided in new code. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
public static extern int gsapi_set_arg_encoding(IntPtr instance,
                                                int encoding);
```

gsapi_set_default_device_list

Set the string containing the list of default device names, for example "display x11alpha x11 bbox". Allows the calling application to influence which device(s) Ghostscript will try, in order, in its selection of the default device. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
public static extern int gsapi_set_default_device_list(IntPtr instance,
                                                       IntPtr list,
                                                      ref int listlen);
```

gsapi_get_default_device_list

Returns a pointer to the current default device string. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
public static extern int gsapi_get_default_device_list(IntPtr instance,
                                                      ref IntPtr list,
                                                         ref int listlen);
```

gsapi_init_with_args

To initialise the interpreter, pass your `instance` of Ghostscript, your argument count, `argc`, and your argument variables, `argv`.

```
public static extern int gsapi_init_with_args(IntPtr instance,
                                              int argc,
                                            IntPtr argv);
```

gsapi_run_*

If these functions return `<= -100`, either quit or a fatal error has occured. You must call gsapi_exit next. The only exception is gsapi_run_string_continue which will return `gs_error_NeedInput` if all is well.

There is a 64 KB length limit on any buffer submitted to a `gsapi_run_*` function for processing. If you have more than 65535 bytes of input then you must split it into smaller pieces and submit each in a separate gsapi_run_string_continue call.

gsapi_run_string_begin

```
public static extern int gsapi_run_string_begin(IntPtr instance,
                                                int usererr,
                                             ref int exitcode);
```

gsapi_run_string_continue

```
public static extern int gsapi_run_string_continue(IntPtr instance,
                                                   IntPtr command,
                                                      int count,
                                                      int usererr,
                                                  ref int exitcode);
```

gsapi_run_string_with_length

```
public static extern int gsapi_run_string_with_length(IntPtr instance,
                                                      IntPtr command,
                                                        uint length,
                                                         int usererr,
                                                     ref int exitcode);
```

gsapi_run_string

```
public static extern int gsapi_run_string(IntPtr instance,
                                          IntPtr command,
                                             int usererr,
                                         ref int exitcode);
```

gsapi_run_string_end

```
public static extern int gsapi_run_string_end(IntPtr instance,
                                                 int usererr,
                                             ref int exitcode);
```

gsapi_run_file

```
public static extern int gsapi_run_file(IntPtr instance,
                                        IntPtr filename,
                                           int usererr,
                                       ref int exitcode);
```

gsapi_exit

Exit the interpreter. This must be called on shutdown if gsapi_init_with_args has been called, and just before gsapi_delete_instance.

```
public static extern int gsapi_exit(IntPtr instance);
```

gsapi_set_param

Sets a parameter.

Broadly, this is equivalent to setting a parameter using -d, -s or -p on the command line. This call cannot be made during a gsapi_run_string operation.

Parameters in this context are not the same as 'arguments' as processed by gsapi_init_with_args, but often the same thing can be achieved. For example, with gsapi_init_with_args, we can pass "-r200" to change the resolution. Broadly the same thing can be achieved by using gsapi_set_param to set a parsed value of "<>".

Internally, when we set a parameter, we perform an `initgraphics` operation. This means that using gsapi_set_param other than at the start of a page is likely to give unexpected results.

Attempting to set a parameter that the device does not recognise will be silently ignored, and that parameter will not be found in subsequent gsapi_get_param calls.

```
public static extern int gsapi_set_param(IntPtr instance,
                                         IntPtr param,
                                         IntPtr value,
                                gs_set_param_type type);
```

> **NOTE**
>
> The `type` argument, as a gs_set_param_type, dictates the kind of object that the `value` argument points to.

> **NOTE**
>
> For more on the C implementation of parameters see: Ghostscript parameters in C.

gsapi_get_param

Retrieve the current value of a parameter.

If an error occurs, the return value is negative. Otherwise the return value is the number of bytes required for storage of the value. Call once with value `NULL` to get the number of bytes required, then call again with value pointing to at least the required number of bytes where the value will be copied out. Note that the caller is required to know the type of value in order to get it. For all types other than gs_spt_string, gs_spt_name, and gs_spt_parsed knowing the type means you already know the size required.

This call retrieves parameters/values that have made it to the device. Thus, any values set using gs_spt_more_to_come without a following call omitting that flag will not be retrieved. Similarly, attempting to get a parameter before gsapi_init_with_args has been called will not list any, even if gsapi_set_param has been used.

Attempting to read a parameter that is not set will return `gs_error_undefined` (-21). Note that calling gsapi_set_param followed by gsapi_get_param may not find the value, if the device did not recognise the key as being one of its configuration keys.

For the C documentation please refer to Ghostscript get_param.

```
public static extern int gsapi_get_param(IntPtr instance,
                                         IntPtr param,
                                         IntPtr value,
                                gs_set_param_type type);
```

gsapi_enumerate_params

Enumerate the current parameters. Call repeatedly to list out the current parameters.

The first call should have `iter` = NULL. Subsequent calls should pass the same pointer in so the iterator can be updated. Negative return codes indicate error, 0 success, and 1 indicates that there are no more keys to read. On success, key will be

updated to point to a null terminated string with the key name that is guaranteed to be valid until the next call to gsapi_enumerate_params. If `type` is non NULL then the pointer `type` will be updated to have the `type` of the parameter.

> **NOTE**
>
> Only one enumeration can happen at a time. Starting a second enumeration will reset the first.

The enumeration only returns parameters/values that have made it to the device. Thus, any values set using the gs_spt_more_to_come without a following call omitting that flag will not be retrieved. Similarly, attempting to enumerate parameters before gsapi_init_with_args has been called will not list any, even if gsapi_set_param has been used.

```
public static extern int gsapi_enumerate_params(IntPtr instance,
                                                out IntPtr iter,
                                                out IntPtr key,
                                                    IntPtr type);
```

gsapi_add_control_path

Add a (case sensitive) path to one of the lists of permitted paths for file access.

```
public static extern int gsapi_add_control_path(IntPtr instance,
                                                   int type,
                                                IntPtr path);
```

gsapi_remove_control_path

Remove a (case sensitive) path from one of the lists of permitted paths for file access.

```
public static extern int gsapi_remove_control_path(IntPtr instance,
                                                      int type,
                                                   IntPtr path);
```

gsapi_purge_control_paths

Clear all the paths from one of the lists of permitted paths for file access.

```
public static extern void gsapi_purge_control_paths(IntPtr instance,
                                                       int type);
```

gsapi_activate_path_control

Enable/Disable path control (i.e. whether paths are checked against permitted paths before access is granted).

```
public static extern void gsapi_activate_path_control(IntPtr instance,
                                                        int enable);
```

gsapi_is_path_control_active

Query whether path control is activated or not.

```
public static extern int gsapi_is_path_control_active(IntPtr instance);
```

Callback and Callout prototypes

GSAPI also defines some prototype pointers which are defined as follows.

gs_stdio_handler

```
/* Callback proto for stdio */
public delegate int gs_stdio_handler(IntPtr caller_handle,
                                     IntPtr buffer,
                                        int len);
```

gsPollHandler

```
/* Callback proto for poll function */
public delegate int gsPollHandler(IntPtr caller_handle);
```

gsCallOut

```
/* Callout proto */
public delegate int gsCallOut(IntPtr callout_handle,
                              IntPtr device_name,
                                 int id,
                                 int size,
                              IntPtr data);
```

GhostNET

About

`GhostNET` is the `C#` interface into the `GhostAPI` library developed for Windows systems.

Enums

Tasks

The Ghostscript task type `enum` is used to inform `GhostAPI` of the type of operation which is being requested.

Task	Description
PS_DISTILL	Task associated with converting a PostScript stream to a PDF document
CREATE_XPS	Task associated with outputting a copy of a document to XPS
SAVE_RESULT	Task associated with saving documents
GET_PAGE_COUNT	Task associated with getting the page count of a document
GENERIC	Generic task identifier
DISPLAY_DEV_THUMBS	Display Device task associated with rendering thumbnails
DISPLAY_DEV_NON_PDF	Display Device task associated with non-PDF or non-XPS rendering [1]
DISPLAY_DEV_PDF	Display Device task associated with PDF & XPS rendering [1]
DISPLAY_DEV_RUN_FILE	Display Device task associated with running files

Task types are defined as `GS_Task_t`.

```
public enum GS_Task_t
{
    PS_DISTILL,
    CREATE_XPS,
    SAVE_RESULT,
    GET_PAGE_COUNT,
    GENERIC,
    DISPLAY_DEV_THUMBS,
    DISPLAY_DEV_NON_PDF,
    DISPLAY_DEV_PDF,
    DISPLAY_DEV_RUN_FILE
}
```

Results

Result types are defined as `GS_Result_t`.

```
public enum GS_Result_t
{
    gsOK,
    gsFAILED,
```

```
    gsCANCELLED
}
```

Status

Status is defined as `gsStatus`.

```
public enum gsStatus
{
    GS_READY,
    GS_BUSY,
    GS_ERROR
};
```

The Parameter Struct

The parameter struct `gsParamState_t` allows for bundles of information to be processed by Ghostscript to complete overall requests.

```
public struct gsParamState_t
{
    public String outputfile;
    public String inputfile;
    public GS_Task_t task;
    public GS_Result_t result;
    public int num_pages;
    public List<int> pages;
    public int firstpage;
    public int lastpage;
    public int currpage;
    public List<String> args;
    public int return_code;
    public double zoom;
    public bool aa;
    public bool is_valid;
};
```

Parameters explained

Setting up your parameters (with any dedicated bespoke method(s) which your application requires) is needed when communicating directly with `GhostAPI`.

When requesting Ghostscript to process an operation an application developer should send a parameter payload which defines the details for the operation.

For example in `GhostNET` we can see the public method as follows:

```
public gsStatus DistillPS(String fileName, int resolution)
{
    gsParamState_t gsparams = new gsParamState_t();
    gsparams.args = new List<string>();

    gsparams.inputfile = fileName;
    gsparams.args.Add("gs");
    gsparams.args.Add("-sDEVICE=pdfwrite");
    gsparams.outputfile = Path.GetTempFileName();
    gsparams.args.Add("-o" + gsparams.outputfile);
    gsparams.task = GS_Task_t.PS_DISTILL;
```

```
    return RunGhostscriptAsync(gsparams);
}
```

Here we can see a parameter payload being setup before being passed on to the asynchronous method `RunGhostscriptAsync` which sets up a worker thread to run according to the task type in the payload.

`GhostNET` handles many common operations on an application developer's behalf, however if you require to write your own methods to interface with `GhostAPI` then referring to the public methods in `GhostNET` is a good starting point.

For full documentation on parameters refer to Ghostscript parameters.

The Event class

`GhostNET` contains a public class `gsEventArgs` which is an extension of the C# class EventArgs. This class is used to set and get events as they occur. `GhostNET` will create these payloads and deliver them back to the application layer's `ProgressCallBack` method asynchronously.

```
public class gsEventArgs : EventArgs
{
    private bool m_completed;
    private int m_progress;
    private gsParamState_t m_param;
    public bool Completed
    {
        get { return m_completed; }
    }
    public gsParamState_t Params
    {
        get { return m_param; }
    }
    public int Progress
    {
        get { return m_progress; }
    }
    public gsEventArgs(bool completed, int progress, gsParamState_t param)
    {
        m_completed = completed;
        m_progress = progress;
        m_param = param;
    }
}
```

GSNET

This class should be instantiated as a member variable in your application with callback definitions setup as required.

Handlers for asynchronous operations can injected by providing your own bespoke callback methods to your instance's `ProgressCallBack` function.

```
/* Set up ghostscript with callbacks for system updates */
m_ghostscript = new GSNET();
m_ghostscript.ProgressCallBack += new GSNET.Progress(gsProgress);
m_ghostscript.StdIOCallBack += new GSNET.StdIO(gsIO);
m_ghostscript.DLLProblemCallBack += new GSNET.DLLProblem(gsDLL);
m_ghostscript.PageRenderedCallBack += new GSNET.PageRendered(gsPageRendered);
m_ghostscript.DisplayDeviceOpen();

/* example callback stubs for asynchronous operations */
private void gsProgress(gsEventArgs asyncInformation)
{
    Console.WriteLine($"gsProgress().progress:{asyncInformation.Progress}");
```

```
if (asyncInformation.Completed) // task complete
{
    // what was the task?
    switch (asyncInformation.Params.task)
    {
        case GS_Task_t.CREATE_XPS:
            Console.WriteLine($"CREATE_XPS.outputfile:");
            Console.WriteLine($"{asyncInformation.Params.result.outputfile}");
            break;

        case GS_Task_t.PS_DISTILL:
            Console.WriteLine($"PS_DISTILL.outputfile:");
            Console.WriteLine($"{asyncInformation.Params.result.outputfile}");
            break;

        case GS_Task_t.SAVE_RESULT:

            break;

        case GS_Task_t.DISPLAY_DEV_THUMBS:

            break;

        case GS_Task_t.DISPLAY_DEV_RUN_FILE:

            break;

        case GS_Task_t.DISPLAY_DEV_PDF:

            break;

        case GS_Task_t.DISPLAY_DEV_NON_PDF:

            break;

        default:

            break;
    }

    // task failed
    if (asyncInformation.Params.result == GS_Result_t.gsFAILED)
    {
        switch (asyncInformation.Params.task)
        {
            case GS_Task_t.CREATE_XPS:

                break;

            case GS_Task_t.PS_DISTILL:

                break;

            case GS_Task_t.SAVE_RESULT:

                break;

            default:

                break;
        }
        return;
    }

    // task cancelled
    if (asyncInformation.Params.result == GS_Result_t.gsCANCELLED)
    {

    }
}
else // task is still running
{
    switch (asyncInformation.Params.task)
```

```
        {
            case GS_Task_t.CREATE_XPS:

                break;

            case GS_Task_t.PS_DISTILL:

                break;

            case GS_Task_t.SAVE_RESULT:

                break;
        }
    }
}

private void gsIO(String message, int len)
{
    Console.WriteLine($"gsIO().message:{message}, length:{len}");
}

private void gsDLL(String message)
{
    Console.WriteLine($"gsDLL().message:{message}");
}

private void gsPageRendered(int width,
                            int height,
                            int raster,
                            IntPtr data,
                            gsParamState_t state)
{

};
```

> **NOTE**
>
> Once a Ghostscript operation is in progress any defined callback functions will be called as the operation runs up unto completion. These callback methods are essential for your application to interpret activity events and react accordingly.

An explanation of callbacks and the available public methods within `GSNET` are explained below.

Delegates

To handle asynchronous events `GhostNET` has four delegates which define callback methods that an application can assign to.

Callback	Description
`DLLProblemCallBack`	Occurs if there is some issue with the Ghostscript DLL
`StdIOCallBack`	Occurs if Ghostscript outputs something to `stderr` or `stdout`
`ProgressCallBack`	Occurs as Ghostscript makes its way through a file
`PageRenderedCallBack`	Occurs when a page has been rendered and the data from the display device is ready

DLLProblemCallBack

```
internal delegate void DLLProblem(String mess);
internal event DLLProblem DLLProblemCallBack;
```

StdIOCallBack

```
internal delegate void StdIO(String mess,
                             int len);
internal event StdIO StdIOCallBack;
```

ProgressCallBack

```
internal delegate void Progress(gsEventArgs info);
internal event Progress ProgressCallBack;
```

PageRenderedCallBack

```
internal delegate void PageRendered(int width,
                                    int height,
                                    int raster,
                                 IntPtr data,
                           gsParamState_t state);
internal event PageRendered PageRenderedCallBack;
```

GetVersion

Use this method to get Ghostscript version info as a handy `String`.

```
public String GetVersion()
```

```
String gs_vers = m_ghostscript.GetVersion();
```

> **NOTE**
>
> An exception will be thrown if there is any issue with the Ghostscript DLL.

DisplayDeviceOpen

Sets up the display device ahead of time.

```
public gsParamState_t DisplayDeviceOpen()
```

```
m_ghostscript.DisplayDeviceOpen();
```

> **NOTE**
>
> Calling this method instantiates ghostscript and configures the encoding and the callbacks for the display device.

DisplayDeviceClose

Closes the display device and deletes the instance.

```
public gsParamState_t DisplayDeviceClose()
```

```
m_ghostscript.DisplayDeviceClose();
```

> **NOTE**
>
> Calling this method deletes ghostscript.

GetPageCount

Use this method to get the number of pages in a supplied document.

```
public int GetPageCount(String fileName)
```

```
int page_number = m_ghostscript.GetPageCount("my_document.pdf");
```

> **NOTE**
>
> If Ghostscript is unable to determine the page count then this method will return `-1`.

CreateXPS

Launches a thread to create an XPS document for Windows printing. This method is asynchronous and logic should be hooked into your application upon GSNET instantiation to interpret progress.

```
public gsStatus CreateXPS(String fileName,
                             int resolution,
                             int num_pages,
                          double width,
                          double height,
                            bool fit_page,
                             int firstpage,
                             int lastpage)
```

```
m_ghostscript.CreateXPS("my_document.pdf",
                        300,
                        10,
                        1000,
                        1000,
                        true,
                        0,
                        9);
```

asynchronous

DistillPS

Launches a thread rendering all the pages of a supplied PostScript file to a PDF.

```
public gsStatus DistillPS(String fileName, int resolution)
```

```
m_ghostscript.DistillPS("my_postscript_document.ps", 300);
```

asynchronous

DisplayDeviceRunFile

Launches a thread to run a file with the display device.

```
public gsStatus DisplayDeviceRunFile(String fileName,
                                     double zoom,
                                       bool aa, // anti-aliasing value
                                        int firstpage,
                                        int lastpage)
```

```
m_ghostscript.DisplayDeviceRunFile("my_document.pdf",
                                   1.0,
                                   true,
                                   0,
                                   9);
```

asynchronous

DisplayDeviceRenderThumbs

Launches a thread rendering all the pages with the display device to collect thumbnail images.

Recommended zoom level for thumbnails is between 0.05 and 0.2, additionally anti-aliasing is probably not required.

```
public gsStatus DisplayDeviceRenderThumbs(String fileName,
                                          double zoom,
                                            bool aa)
```

```
m_ghostscript.DisplayDeviceRenderThumbs("my_document.pdf",
                                        0.1,
                                        false);
```

asynchronous

DisplayDeviceRenderPages

Launches a thread rendering a set of pages with the display device. For use with languages that can be indexed via pages which include PDF and XPS. [1]

```
public gsStatus DisplayDeviceRenderPages(String fileName,
                                            int first_page,
                                            int last_page,
                                         double zoom)
```

```
m_ghostscript.DisplayDeviceRenderPages("my_document.pdf",
                                       0,
                                       9,
                                       1.0);
```

asynchronous

GetStatus

Returns the current status of `Ghostscript`.

```
public gsStatus GetStatus()
```

```
gsStatus status = m_ghostscript.GetStatus();
```

Cancel

Cancels asynchronous operations.

```
public void Cancel()
```

```
m_ghostscript.Cancel();
```

GhostscriptException

An application developer can log any exceptions in this public class as required by editing the constructor.

```
public class GhostscriptException : Exception
{
    public GhostscriptException(string message) : base(message)
    {
        // Report exceptions as required
    }
}
```

Notes

1: Ghostscript & Page Description Languages

Ghostscript handles the following PDLs: `PCL` `PDF` `PS` `XPS`.

`PCL` and `PS` do not allow random access, meaning that, to print page 2 in a 100 page document, Ghostscript has to read the entire document stream of 100 pages.

On the other hand, `PDF` and `XPS` allow for going directly to page 2 and then only dealing with that content. The tasks `DISPLAY_DEV_NON_PDF` and `DISPLAY_DEV_PDF` keep track of what sort of input Ghostscript is dealing with and enables the

application to direct progress or completion callbacks accordingly.

GhostMono

About

`GhostMono` is the `C#` interface into the `GhostAPI` library developed for Linux systems.

Enums

Tasks

The Ghostscript task type `enum` is used to inform `GhostAPI` of the type of operation which is being requested.

Task	Description
PS_DISTILL	Task associated with converting a PostScript stream to a PDF document
CREATE_XPS	Task associated with outputting a copy of a document to XPS
SAVE_RESULT	Task associated with saving documents
GET_PAGE_COUNT	Task associated with getting the page count of a document
GENERIC	Generic task identifier
DISPLAY_DEV_THUMBS	Display Device task associated with rendering thumbnails
DISPLAY_DEV_NON_PDF	Display Device task associated with non-PDF or non-XPS rendering [1]
DISPLAY_DEV_PDF	Display Device task associated with PDF & XPS rendering [1]
DISPLAY_DEV_RUN_FILE	Display Device task associated with running files

Task types are defined as `GS_Task_t`.

```
public enum GS_Task_t
{
    PS_DISTILL,
    CREATE_XPS,
    SAVE_RESULT,
    GET_PAGE_COUNT,
    GENERIC,
    DISPLAY_DEV_THUMBS,
    DISPLAY_DEV_NON_PDF,
    DISPLAY_DEV_PDF,
    DISPLAY_DEV_RUN_FILE
}
```

Results

Result types are defined as `GS_Result_t`.

```
public enum GS_Result_t
{
    gsOK,
    gsFAILED,
```

```
    gsCANCELLED
}
```

Status

Status is defined as `gsStatus`.

```
public enum gsStatus
{
    GS_READY,
    GS_BUSY,
    GS_ERROR
};
```

The Parameter Struct

The parameter struct `gsParamState_t` allows for bundles of information to be processed by Ghostscript to complete overall requests.

```
public struct gsParamState_t
{
    public String outputfile;
    public String inputfile;
    public GS_Task_t task;
    public GS_Result_t result;
    public int num_pages;
    public List<int> pages;
    public int firstpage;
    public int lastpage;
    public int currpage;
    public List<String> args;
    public int return_code;
    public double zoom;
};
```

Parameters explained

Setting up your parameters (with any dedicated bespoke method(s) which your application requires) is needed when communicating directly with `GhostAPI`.

When requesting Ghostscript to process an operation an application developer should send a parameter payload which defines the details for the operation.

For example in `GhostMono` we can see the public method as follows:

```
public gsStatus DistillPS(String fileName, int resolution)
{
    gsParamState_t gsparams = new gsParamState_t();
    gsparams.args = new List<string>();

    gsparams.inputfile = fileName;
    gsparams.args.Add("gs");
    gsparams.args.Add("-dNOPAUSE");
    gsparams.args.Add("-dBATCH");
    gsparams.args.Add("-I%rom%Resource/Init/");
    gsparams.args.Add("-dSAFER");
    gsparams.args.Add("-sDEVICE=pdfwrite");
    gsparams.outputfile = Path.GetTempFileName();
    gsparams.args.Add("-o" + gsparams.outputfile);
```

```
    gsparams.task = GS_Task_t.PS_DISTILL;

    return RunGhostscriptAsync(gsparams);
}
```

Here we can see a parameter payload being setup before being passed on to the asynchronous method `RunGhostscriptAsync` which sets up a worker thread to run according to the task type in the payload.

`GhostMono` handles many common operations on an application developer's behalf, however if you require to write your own methods to interface with `GhostAPI` then referring to the public methods in `GhostMono` is a good starting point.

For full documentation on parameters refer to Ghostscript parameters.

The Event class

`GhostMono` contains a public class `gsThreadCallBack`. This class is used to set and get callback information as they occur. `GhostMono` will create these payloads and deliver them back to the application layer's `ProgressCallBack` method asynchronously.

```
public class gsThreadCallBack
{
    private bool m_completed;
    private int m_progress;
    private gsParamState_t m_param;
    public bool Completed
    {
        get { return m_completed; }
    }
    public gsParamState_t Params
    {
        get { return m_param; }
    }
    public int Progress
    {
        get { return m_progress; }
    }
    public gsThreadCallBack(bool completed, int progress, gsParamState_t param)
    {
        m_completed = completed;
        m_progress = progress;
        m_param = param;
    }
}
```

GSMONO

This class should be instantiated as a member variable in your application with callback definitions setup as required.

Handlers for asynchronous operations can injected by providing your own bespoke callback methods to your instance's `ProgressCallBack` function.

```
/* Set up ghostscript with callbacks for system updates */
m_ghostscript = new GSMONO();
m_ghostscript.ProgressCallBack += new GSMONO.Progress(gsProgress);
m_ghostscript.StdIOCallBack += new GSMONO.StdIO(gsIO);
m_ghostscript.DLLProblemCallBack += new GSMONO.DLLProblem(gsDLL);
m_ghostscript.PageRenderedCallBack += new GSMONO.PageRendered(gsPageRendered);
m_ghostscript.DisplayDeviceOpen();

/* example callback stubs for asynchronous operations */
private void gsProgress(gsThreadCallBack asyncInformation)
```

```
{
    Console.WriteLine($"gsProgress().progress:{asyncInformation.Progress}");

    if (asyncInformation.Completed) // task complete
    {
        // what was the task?
        switch (asyncInformation.Params.task)
        {
            case GS_Task_t.CREATE_XPS:
                Console.WriteLine($"CREATE_XPS.outputfile:");
                Console.WriteLine($"{asyncInformation.Params.result.outputfile}");
                break;

            case GS_Task_t.PS_DISTILL:
                Console.WriteLine($"PS_DISTILL.outputfile:");
                Console.WriteLine($"{asyncInformation.Params.result.outputfile}");
                break;

            case GS_Task_t.SAVE_RESULT:

                break;

            case GS_Task_t.DISPLAY_DEV_THUMBS:

                break;

            case GS_Task_t.DISPLAY_DEV_RUN_FILE:

                break;

            case GS_Task_t.DISPLAY_DEV_PDF:

                break;

            case GS_Task_t.DISPLAY_DEV_NON_PDF:

                break;

            default:

                break;
        }

        // task failed
        if (asyncInformation.Params.result == GS_Result_t.gsFAILED)
        {
            switch (asyncInformation.Params.task)
            {
                case GS_Task_t.CREATE_XPS:

                    break;

                case GS_Task_t.PS_DISTILL:

                    break;

                case GS_Task_t.SAVE_RESULT:

                    break;

                default:

                    break;
            }
            return;
        }

        // task cancelled
        if (asyncInformation.Params.result == GS_Result_t.gsCANCELLED)
        {

        }
    }
    else // task is still running
```

```
    {
        switch (asyncInformation.Params.task)
        {
            case GS_Task_t.CREATE_XPS:

                break;

            case GS_Task_t.PS_DISTILL:

                break;

            case GS_Task_t.SAVE_RESULT:

                break;
        }
    }
}

private void gsIO(String message, int len)
{
    Console.WriteLine($"gsIO().message:{message}, length:{len}");
}

private void gsDLL(String message)
{
    Console.WriteLine($"gsDLL().message:{message}");
}

private void gsPageRendered(int width,
                            int height,
                            int raster,
                            IntPtr data,
                            gsParamState_t state)
{

};
```

> **NOTE**
>
> Once a Ghostscript operation is in progress any defined callback functions will be called as the operation runs up unto completion. These callback methods are essential for your application to interpret activity events and react accordingly.

An explanation of callbacks and the available public methods within `GSMONO` are explained below.

Delegates

To handle asynchronous events `GhostMONO` has four delegates which define callback methods that an application can assign to.

Callback	Description
`DLLProblemCallBack`	Occurs if there is some issue with the Ghostscript Shared Object (`libgpdl.so`)
`StdIOCallBack`	Occurs if Ghostscript outputs something to `stderr` or `stdout`
`ProgressCallBack`	Occurs as Ghostscript makes its way through a file
`PageRenderedCallBack`	Occurs when a page has been rendered and the data from the display device is ready

DLLProblemCallBack

```
internal delegate void DLLProblem(String mess);
internal event DLLProblem DLLProblemCallBack;
```

StdIOCallBack

```
internal delegate void StdIO(String mess,
                             int len);
internal event StdIO StdIOCallBack;
```

ProgressCallBack

```
internal delegate void Progress(gsEventArgs info);
internal event Progress ProgressCallBack;
```

PageRenderedCallBack

```
internal delegate void PageRendered(int width,
                                    int height,
                                    int raster,
                                 IntPtr data,
                           gsParamState_t state);
internal event PageRendered PageRenderedCallBack;
```

GetVersion

Use this method to get Ghostscript version info as a handy `String`.

```
public String GetVersion()
```

```
String gs_vers = m_ghostscript.GetVersion();
```

> **NOTE**
>
> An exception will be thrown if there is any issue with the Ghostscript DLL.

DisplayDeviceOpen

Sets up the display device ahead of time.

```
public gsParamState_t DisplayDeviceOpen()
```

```
m_ghostscript.DisplayDeviceOpen();
```

> **NOTE**
>
> Calling this method instantiates ghostscript and configures the encoding and the callbacks for the display device.

DisplayDeviceClose

Closes the display device and deletes the instance.

```
public gsParamState_t DisplayDeviceClose()
```

```
m_ghostscript.DisplayDeviceClose();
```

> **NOTE**
>
> Calling this method deletes ghostscript.

GetPageCount

Use this method to get the number of pages in a supplied document.

```
public int GetPageCount(String fileName)
```

```
int page_number = m_ghostscript.GetPageCount("my_document.pdf");
```

> **NOTE**
>
> If Ghostscript is unable to determine the page count then this method will return `-1`.

DistillPS

Launches a thread rendering all the pages of a supplied PostScript file to a PDF.

```
public gsStatus DistillPS(String fileName, int resolution)
```

```
m_ghostscript.DistillPS("my_postscript_document.ps", 300);
```

asynchronous

DisplayDeviceRenderAll

Launches a thread rendering all the document pages with the display device. For use with languages that can be indexed via pages which include PDF and XPS. 1

```
public gsStatus DisplayDeviceRenderAll(String fileName, double zoom, bool aa, GS_Task_t task)
```

```
m_ghostscript.DisplayDeviceRenderAll("my_document.pdf",
                                     0.1,
                                     false,
                                     GS_Task_t.DISPLAY_DEV_THUMBS_NON_PDF);
```

asynchronous

DisplayDeviceRenderThumbs

Launches a thread rendering all the pages with the display device to collect thumbnail images.

Recommended zoom level for thumbnails is between 0.05 and 0.2, additionally anti-aliasing is probably not required.

```
public gsStatus DisplayDeviceRenderThumbs(String fileName,
                                          double zoom,
                                            bool aa)
```

```
m_ghostscript.DisplayDeviceRenderThumbs("my_document.pdf",
                                        0.1,
                                        false);
```

asynchronous

DisplayDeviceRenderPages

Launches a thread rendering a set of pages with the display device. For use with languages that can be indexed via pages which include PDF and XPS. [1]

```
public gsStatus DisplayDeviceRenderPages(String fileName,
                                            int first_page,
                                            int last_page,
                                         double zoom)
```

```
m_ghostscript.DisplayDeviceRenderPages("my_document.pdf",
                                       0,
                                       9,
                                       1.0);
```

asynchronous

GetStatus

Returns the current status of `Ghostscript`.

```
public gsStatus GetStatus()
```

```
gsStatus status = m_ghostscript.GetStatus();
```

GhostscriptException

An application developer can log any exceptions in this public class as required by editing the constructor.

```
public class GhostscriptException : Exception
{
    public GhostscriptException(string message) : base(message)
    {
        // Report exceptions as required
    }
}
```

Notes

1: Ghostscript & Page Description Languages

Ghostscript handles the following PDLs: `PCL` `PDF` `PS` `XPS` .

`PCL` and `PS` do not allow random access, meaning that, to print page 2 in a 100 page document, Ghostscript has to read the entire document stream of 100 pages.

On the other hand, `PDF` and `XPS` allow for going directly to page 2 and then only dealing with that content. The tasks `DISPLAY_DEV_NON_PDF` and `DISPLAY_DEV_PDF` keep track of what sort of input Ghostscript is dealing with and enables the application to direct progress or completion callbacks accordingly.

Java overview

About

In the GhostPDL repository sample `Java` projects can be found in `/demos/java`.

Within this location the following folders are of relevance:

- jni `jni`
- gsjava `gsjava`
- gstest `gstest`
- gsviewer `gsviewer`

Platform & setup

Building Ghostscript

Ghostscript should be built as a shared library for your platform.

See Building Ghostscript.

jni: Building the Java Native Interface

Before building the JNI ensure that Ghostscript has already been built for your platform and that you have JDK installed.

The JNI is for use in the Java interface,
this object must be placed somewhere on your Java PATH. On Windows, the DLL
can be placed in the working directory, next to `gsjava.jar`.

Platform	JNI file
Windows	`gs_jni.dll`
MacOS	`gs_jni.dylib`
Linux / OpenBSD	`gs_jni.so`

Preparing your include folder

The build scripts require the header `jni.h`, which defines
all JNI functions, and `jni_md.h`, which defines all system-specific
integer types. The build scripts expect an include folder relative to their location which contain these header files from your system.

These headers are typically found in the following directories:

Platform	jni.h	jni_md.h
Windows	`C:\Program Files\Java<JDK Install>\include\jni.h`	`C:\Program Files\Java<JDK Install>\include\win32\jni_md.h`
MacOS	`/Library/Java/JavaVirtualMachines/<JDK Install>/Contents/Home/include/jni.h`	`/Library/Java/JavaVirtualMachines/<JDK Install>/Contents/Home/include/darwin/jni_md.h`

Platform	jni.h	jni_md.h
Linux	`/lib/jvm/<JDK Install>/include/jni.h`	`/lib/jvm/<JDK Install>/include/linux/jni_md.h`

Once your `include` folder has been located folder you can copy it and place it in your `ghostpdl/demos/java/jni/gs_jni` folder.

Your build scripts should now be ready to run as they will be able to find the required JNI header files in their own relative include folder.

Building on Windows

The `jni` folder contains a Visual Studio Solution file `/jni/gs_jni/gs_jni.sln` which you should use to build the required JNI `gs_jni.dll` library file.

With the project open in Visual Studio, select the required architecture from the drop down - then right click on 'gs_jni' in the solution explorer and choose "Build".

Building on MacOS

On your command line, navigate to `ghostpdl/demos/java/jni/gs_jni` and ensure that the build script is executable and then run it, with:

```
chmod +x build_darwin.sh
./build_darwin.sh
```

Building on Linux

On your command line, navigate to `ghostpdl/demos/java/jni/gs_jni` and ensure that the build script is executable and then run it, with:

```
chmod +x build_linux.sh
./build_linux.sh
```

gsjava: Building the JAR

Building with the command line

Navigate to `ghostpdl/demos/java/gsjava` and use the following:

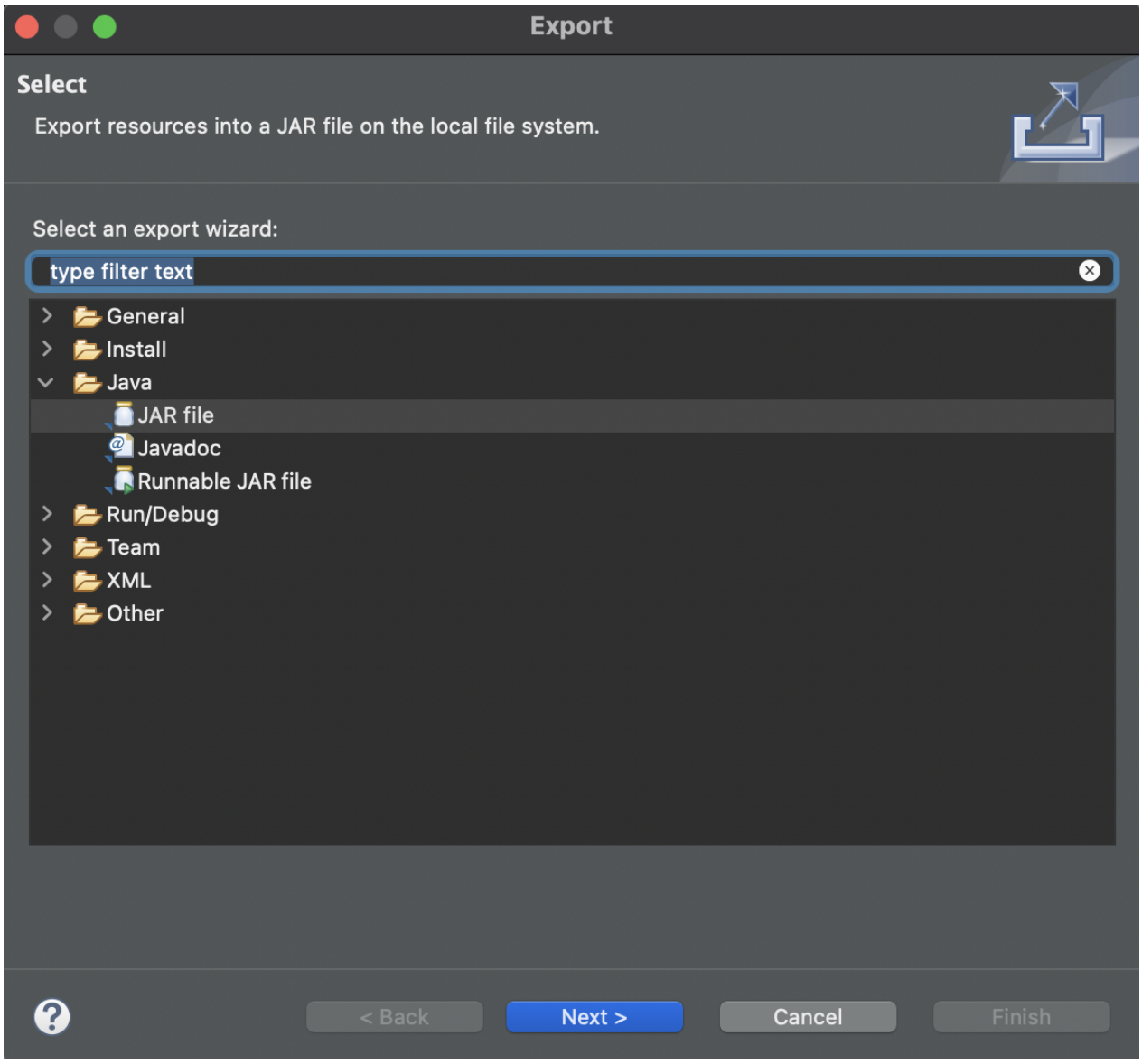
Platform	Run file
Windows	`build_win32.bat`
MacOS	`build_darwin.sh`
Linux	`build_linux.sh`

> **NOTE**
>
> `gsjava` has a dependancy on jni, please ensure that `gs_jni` is able to be built beforehand.

Building with Eclipse

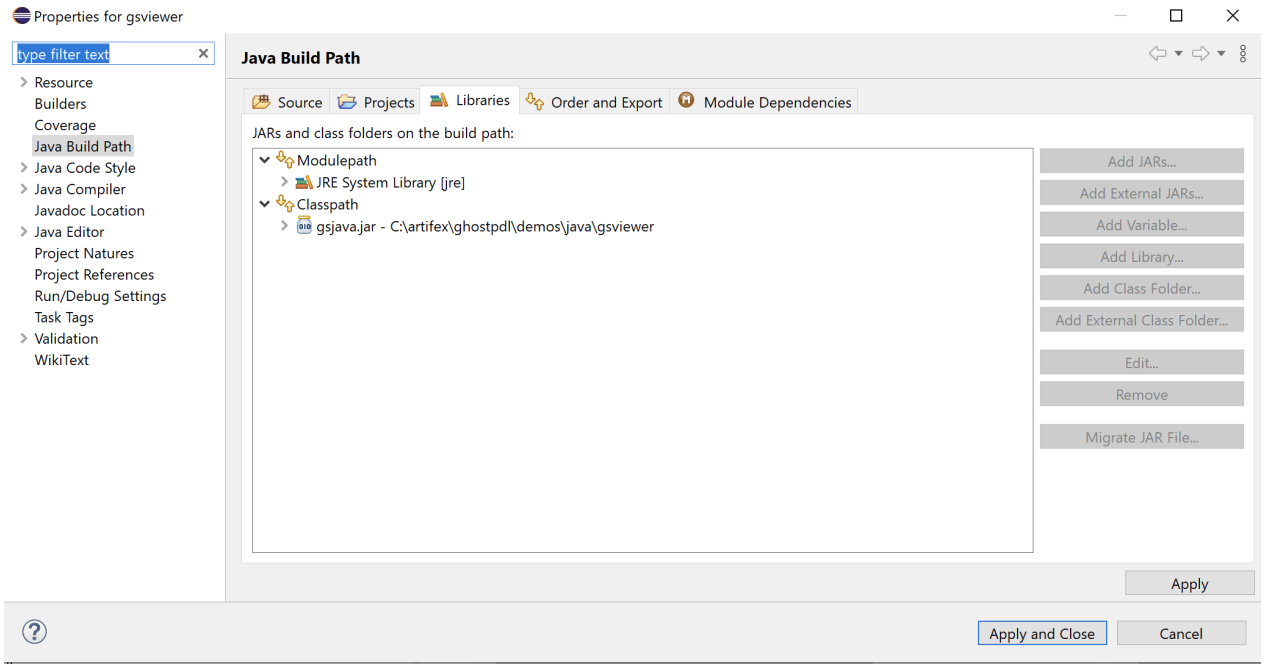
Alternatively you can use Eclipse to build the JAR file.

Using Eclipse import the source folder `gsjava` as a project and select `Export > Java > JAR File` as shown in the screenshot example below:



Linking the JAR

The built JAR should be properly linked within your project Java Build Path as follows:



Demo projects

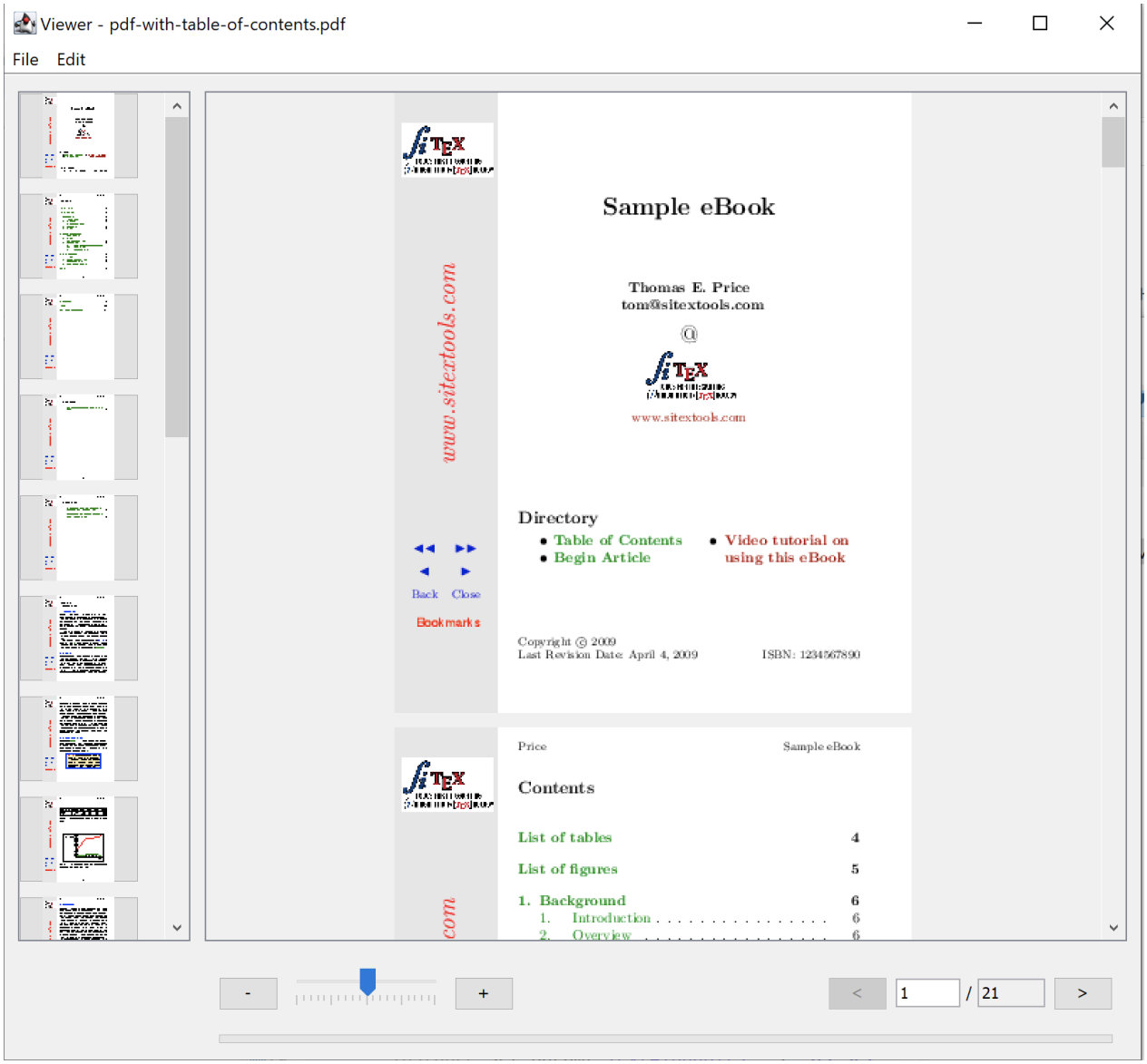
gstest

This project can be opened in Eclipse and used to test the Ghostscript API. The sample here simply sets up an instance of Ghostscript and then sets and gets some parameters accordingly.

gsviewer

This project can be used to test the Ghostscript API alongside a UI which handles opening PostScript and PDF files. The sample application here allows for file browsing and Ghostscript file viewing.

Below is a screenshot of the sample application with a PDF open:



To run the project navigate to the `demos/java/gsviewer` location and ensure that the required libraries are in the directory:

Platform	Ghostscript library file	JNI library file
Windows	`gpdldll64.dll`	`gs_jni.dll`
MacOS	`libgpdl.dylib`	`gs_jni.dylib`
Linux / OpenBSD	`libgpdl.so` (this may have been built as `libgs.so`, so it should be copied into this directory and renamed to `libgpdl.so`)	`gs_jni.so`

Building on Windows

Run the `build_win32.bat` script.

Running on Windows

To run, open `gsviewer.jar` either through File Explorer or in the command line through the following command:

```
java -jar gsviewer.jar
```

Building on MacOS

On your command line, navigate to `ghostpdl/demos/java/gsviewer` and ensure that the build script is executable and then run it, with:

```
chmod +x build_darwin.sh
./build_darwin.sh
```

This will automatically build `gs_jni.dylib` (in the `ghostpdl/demos/java/jni/gs_jni/` location) and `gsjava.jar` `gsviewer.jar` in the `gsviewer` directory.

Running on MacOS

Ensure that the Ghostscript library exists in the `gsviewer` directory. (Copy and move the built library from `ghostpdl/sobin` as required).

Ensure that the run script is executable and then run it, with:

```
chmod +x start_darwin.sh
./start_darwin.sh
```

Building on Linux

On your command line, navigate to `ghostpdl/demos/java/gsviewer` and ensure that the build script is executable and then run it, with:

```
chmod +x build_linux.sh
./build_linux.sh
```

This will automatically build `gs_jni.so` (in the `ghostpdl/demos/java/jni/gs_jni/` location) and `gsjava.jar` `gsviewer.jar` in the `gsviewer` directory.

> **NOTE**
>
> On Linux, when using OpenJDK, the property "assistive_technologies" may need to be modified for the Java code to build. It can be modified by editing the "accessibility.properties" file. This is located at:
>
> `/etc/java-8-openjdk/accessibility.properties`

Running on Linux

Ensure that the Ghostscript library exists in the `gsviewer` directory. (Copy and move the built library from `ghostpdl/sobin` as required).

Ensure that the run script is executable and then run it, with:

```
chmod +x start_linux.sh
./start_linux.sh
```

gsjava.jar

About

`gsjava.jar` is the Java library which contains classes and interfaces which enable API calls required to use Ghostscript.

Assuming that the JAR for your project has been built and properly linked with your own project then the Ghostscript API should be available by importing the required classes within your project's `.java` files.

GSAPI & GSInstance

- GSAPI is the main Ghostscript API class which bridges into the Ghostscript C library.
- GSInstance is a wrapper class for GSAPI which encapsulates an instance of Ghostscript and allows for simpler API calls.

```
// to use GSAPI
import static com.artifex.gsjava.GSAPI.*;

// to use GSInstance
import com.artifex.gsjava.GSInstance;
```

GSAPI

gsapi_revision

This method returns the revision numbers and strings of the Ghostscript interpreter library; you should call it before any other interpreter library functions to make sure that the correct version of the Ghostscript interpreter has been loaded.

```
public static native int gsapi_revision(GSAPI.Revision revision,
                                         int len);
```

> **NOTE**
>
> The method should write to a reference variable which conforms to the class GSAPI.Revision.

GSAPI.Revision

This class is used to store information about Ghostscript and provides handy getters for the product and the copyright information.

```
public static class Revision {
    public volatile byte[] product;
    public volatile byte[] copyright;
    public volatile long revision;
    public volatile long revisionDate;

    public Revision() {
        this.product = null;
        this.copyright = null;
        this.revision = 0L;
        this.revisionDate = 0L;
    }

    /**
     * Returns the product information as a String.
```

```
     *
     * @return The product information.
     */
    public String getProduct() {
        return new String(product);
    }

    /**
     * Returns the copyright information as a String.
     *
     * @return The copyright information.
     */
    public String getCopyright() {
        return new String(copyright);
    }
}
```

gsapi_new_instance

Creates a new instance of Ghostscript. This instance is passed to most other GSAPI methods. Unless Ghostscript has been compiled with the `GS_THREADSAFE` define, only one instance at a time is supported.

```
public static native int gsapi_new_instance(Reference<Long> instance,
                                             long callerHandle);
```

> **NOTE**
>
> The method returns a reference which represents your instance of Ghostscript.

gsapi_delete_instance

Destroy an instance of Ghostscript. Before you call this, Ghostscript must have finished. If Ghostscript has been initialised, you should call gsapi_exit beforehand.

```
public static native void gsapi_delete_instance(long instance);
```

gsapi_set_stdio_with_handle

Set the callback functions for `stdio`, together with the handle to use in the callback functions. The `stdin` callback function should return the number of characters read, 0 for EOF, or -1 for error. The `stdout` and `stderr` callback functions should return the number of characters written.

```
public static native int gsapi_set_stdio_with_handle(long instance,
                                          IStdInFunction stdin,
                                         IStdOutFunction stdout,
                                         IStdErrFunction stderr,
                                                    long callerHandle);
```

gsapi_set_stdio

Set the callback functions for `stdio`. The handle used in the callbacks will be taken from the value passed to gsapi_new_instance. Otherwise the behaviour of this function matches gsapi_set_stdio_with_handle.

```
public static native int gsapi_set_stdio(long instance,
                                 IStdInFunction stdin,
```

```
                                IStdOutFunction stdout,
                                IStdErrFunction stderr);
```

gsapi_set_poll_with_handle

Set the callback function for polling, together with the handle to pass to the callback function. This function will only be called if the Ghostscript interpreter was compiled with `CHECK_INTERRUPTS` as described in `gpcheck.h`.

The polling function should return zero if all is well, and return negative if it wants ghostscript to abort. This is often used for checking for a user cancel. This can also be used for handling window events or cooperative multitasking.

The polling function is called very frequently during interpretation and rendering so it must be fast. If the function is slow, then using a counter to `return 0` immediately some number of times can be used to reduce the performance impact.

```
public static native int gsapi_set_poll_with_handle(long instance,
                                          IPollFunction pollfun,
                                                  long callerHandle);
```

```

```

gsapi_set_poll

Set the callback function for polling. The handle passed to the callback function will be taken from the handle passed to gsapi_new_instance. Otherwise the behaviour of this function matches gsapi_set_poll_with_handle.

```
public static native int gsapi_set_poll(long instance,
                                IPollFunction pollfun);
```

gsapi_set_display_callback

This call is deprecated; please use gsapi_register_callout to register a callout handler for the display device in preference.

```
public static native int gsapi_set_display_callback(long instance,
                                       DisplayCallback displayCallback);
```

gsapi_register_callout

This call registers a callout handler.

```
public static native int gsapi_register_callout(long instance,
                                    ICalloutFunction callout,
                                                long calloutHandle);
```

gsapi_deregister_callout

This call deregisters a callout handler previously registered with gsapi_register_callout. All three arguments must match exactly for the callout handler to be deregistered.

```
public static native void gsapi_deregister_callout(long instance,
                                    ICalloutFunction callout,
                                                long calloutHandle);
```

gsapi_set_arg_encoding

Set the encoding used for the interpretation of all subsequent arguments supplied via the `GSAPI` interface on this instance. By default we expect args to be in encoding `0` (the 'local' encoding for this OS). On Windows this means "the currently selected codepage". This means that omitting to call this function will leave Ghostscript running exactly as it always has. Please note that use of the 'local' encoding is now deprecated and should be avoided in new code. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
public static native int gsapi_set_arg_encoding(long instance,
                                            int encoding);
```

gsapi_set_default_device_list

Set the string containing the list of default device names, for example "display x11alpha x11 bbox". Allows the calling application to influence which device(s) Ghostscript will try, in order, in its selection of the default device. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
public static native int gsapi_set_default_device_list(long instance,
                                                byte[] list,
                                                   int listlen);
```

gsapi_get_default_device_list

Returns a pointer to the current default device string. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
public static native int gsapi_get_default_device_list(long instance,
                                        Reference<byte[]> list,
                                       Reference<Integer> listlen);
```

gsapi_init_with_args

To initialise the interpreter, pass your `instance` of Ghostscript, your argument count: `argc`, and your argument variables: `argv`.

```
public static native int gsapi_init_with_args(long instance,
                                            int argc,
                                         byte[][] argv);
```

> **NOTE**
>
> There are also simpler utility methods which eliminates the need to send through your argument count and allows for simpler `String` passing for your argument array.

Utility methods:

```
public static int gsapi_init_with_args(long instance,
                                       String[] argv);
```

```
public static int gsapi_init_with_args(long instance,
                                List<String> argv);
```

gsapi_run_*

If these functions return `<= -100`, either quit or a fatal error has occured. You must call gsapi_exit next. The only exception is gsapi_run_string_continue which will return `gs_error_NeedInput` if all is well.

There is a 64 KB length limit on any buffer submitted to a `gsapi_run_*` function for processing. If you have more than 65535 bytes of input then you must split it into smaller pieces and submit each in a separate gsapi_run_string_continue call.

gsapi_run_string_begin

```
public static native int gsapi_run_string_begin(long instance,
                                                 int userErrors,
                                    Reference<Integer> pExitCode);
```

gsapi_run_string_continue

```
public static native int gsapi_run_string_continue(long instance,
                                                   byte[] str,
                                                      int length,
                                                      int userErrors,
                                       Reference<Integer> pExitCode);
```

> **NOTE**
>
> There is a simpler utility method which allows for simpler `String` passing for the `str` argument.

Utility method:

```
public static int gsapi_run_string_continue(long instance,
                                            String str,
                                               int length,
                                               int userErrors,
                                 Reference<Integer> pExitCode);
```

gsapi_run_string_with_length

```
public static native int gsapi_run_string_with_length(long instance,
                                                      byte[] str,
                                                         int length,
                                                         int userErrors,
                                          Reference<Integer> pExitCode);
```

> **NOTE**

> There is a simpler utility method which allows for simpler `String` passing for the `str` argument.

Utility method:

```
public static int gsapi_run_string_with_length(long instance,
                                               String str,
                                                  int length,
                                                  int userErrors,
                                    Reference<Integer> pExitCode);
```

gsapi_run_string

```
public static native int gsapi_run_string(long instance,
                                          byte[] str,
                                             int userErrors,
                                 Reference<Integer> pExitCode);
```

> **NOTE**
>
> There is a simpler utility method which allows for simpler `String` passing for the `str` argument.

Utility method:

```
public static int gsapi_run_string(long instance,
                                   String str,
                                      int userErrors,
                         Reference<Integer> pExitCode);
```

gsapi_run_string_end

```
public static native int gsapi_run_string_end(long instance,
                                              int userErrors,
                                  Reference<Integer> pExitCode);
```

gsapi_run_file

```
public static native int gsapi_run_file(long instance,
                                        byte[] fileName,
                                           int userErrors,
                               Reference<Integer> pExitCode);
```

> **NOTE**
>
> There is a simpler utility method which allows for simpler `String` passing for the `fileName` argument.

Utility method:

```
public static int gsapi_run_file(long instance,
                                 String fileName,
                                    int userErrors,
                       Reference<Integer> pExitCode);
```

gsapi_exit

Exit the interpreter. This must be called on shutdown if gsapi_init_with_args has been called, and just before gsapi_delete_instance.

```
public static native int gsapi_exit(long instance);
```

gsapi_set_param

Sets a parameter. Broadly, this is equivalent to setting a parameter using -d, -s or -p on the command line. This call cannot be made during a gsapi_run_string operation.

Parameters in this context are not the same as 'arguments' as processed by gsapi_init_with_args, but often the same thing can be achieved. For example, with gsapi_init_with_args, we can pass "-r200" to change the resolution. Broadly the same thing can be achieved by using gsapi_set_param to set a parsed value of "<>".

Internally, when we set a parameter, we perform an `initgraphics` operation. This means that using gsapi_set_param other than at the start of a page is likely to give unexpected results.

Attempting to set a parameter that the device does not recognise will be silently ignored, and that parameter will not be found in subsequent gsapi_get_param calls.

```
public static native int gsapi_set_param(long instance,
                                         byte[] param,
                                         Object value,
                                            int paramType);
```

> **NOTE**
>
> The `type` argument, as a gs_set_param_type, dictates the kind of object that the `value` argument points to.

> **NOTE**
>
> For more on the C implementation of parameters see: Ghostscript parameters in C.

> **NOTE**
>
> There are also simpler utility methods which allows for simpler `String` passing for your arguments.

Utility methods:

```
public static int gsapi_set_param(long instance,
                                  String param,
                                  String value,
                                     int paramType);
```

```
public static int gsapi_set_param(long instance,
                                  String param,
                                  Object value,
                                     int paramType);
```

gsapi_get_param

Retrieve the current value of a parameter.

If an error occurs, the return value is negative. Otherwise the return value is the number of bytes required for storage of the value. Call once with value `NULL` to get the number of bytes required, then call again with value pointing to at least the required number of bytes where the value will be copied out. Note that the caller is required to know the type of value in order to get it. For all types other than gs_spt_string, gs_spt_name, and gs_spt_parsed knowing the type means you already know the size required.

This call retrieves parameters/values that have made it to the device. Thus, any values set using gs_spt_more_to_come without a following call omitting that flag will not be retrieved. Similarly, attempting to get a parameter before gsapi_init_with_args has been called will not list any, even if gsapi_set_param has been used.

Attempting to read a parameter that is not set will return `gs_error_undefined` (-21). Note that calling gsapi_set_param followed by gsapi_get_param may not find the value, if the device did not recognise the key as being one of its configuration keys.

For the C documentation please refer to Ghostscript get_param.

```
public static native int gsapi_get_param(long instance,
                                         byte[] param,
                                           long value,
                                            int paramType);
```

> **NOTE**
>
> There is a simpler utility method which allows for simpler `String` passing for the `param` argument.

Utility method:

```
public static int gsapi_get_param(long instance,
                                  String param,
                                    long value,
                                     int paramType);
```

gsapi_enumerate_params

Enumerate the current parameters. Call repeatedly to list out the current parameters.

The first call should have `iter` = NULL. Subsequent calls should pass the same pointer in so the iterator can be updated. Negative return codes indicate error, 0 success, and 1 indicates that there are no more keys to read. On success, key will be updated to point to a null terminated string with the key name that is guaranteed to be valid until the next call to gsapi_enumerate_params. If `type` is non NULL then the pointer `type` will be updated to have the `type` of the parameter.

> **NOTE**
>
> Only one enumeration can happen at a time. Starting a second enumeration will reset the first.

The enumeration only returns parameters/values that have made it to the device. Thus, any values set using the gs_spt_more_to_come without a following call omitting that flag will not be retrieved. Similarly, attempting to enumerate parameters before gsapi_init_with_args has been called will not list any, even if gsapi_set_param has been used.

```
public static native int gsapi_enumerate_params(long instance,
                                       Reference<Long> iter,
                                    Reference<byte[]> key,
                                   Reference<Integer> paramType);
```

gsapi_add_control_path

Add a (case sensitive) path to one of the lists of permitted paths for file access.

```
public static native int gsapi_add_control_path(long instance,
                                                 int type,
                                               byte[] path);
```

> **NOTE**
>
> There is a simpler utility method which allows for simpler `String` passing for the `path` argument.

Utility method:

```
public static int gsapi_add_control_path(long instance,
                                         int type,
                                      String path);
```

gsapi_remove_control_path

Remove a (case sensitive) path from one of the lists of permitted paths for file access.

```
public static native int gsapi_remove_control_path(long instance,
                                                    int type,
                                                  byte[] path);
```

> **NOTE**
>
> There is a simpler utility method which allows for simpler `String` passing for the `path` argument.

Utility method:

```
public static int gsapi_remove_control_path(long instance,
                                            int type,
                                         String path);
```

gsapi_purge_control_paths

Clear all the paths from one of the lists of permitted paths for file access.

```
public static native void gsapi_purge_control_paths(long instance,
                                                     int type);
```

gsapi_activate_path_control

Enable/Disable path control (i.e. whether paths are checked against permitted paths before access is granted).

```
public static native void gsapi_activate_path_control(long instance,
                                                       boolean enable);
```

gsapi_is_path_control_active

Query whether path control is activated or not.

```
public static native boolean gsapi_is_path_control_active(long instance);
```

Callback & Callout interfaces

`gsjava.jar` also defines some functional interfaces for callbacks & callouts with `package com.artifex.gsjava.callback` which are defined as follows.

IStdInFunction

```
public interface IStdInFunction {
    /**
     * @param callerHandle The caller handle.
     * @param buf A string represented by a byte array.
     * @param len The number of bytes to read.
     * @return The number of bytes read, must be <code>len</code>/
     */
    public int onStdIn(long callerHandle,
                       byte[] buf,
                          int len);
}
```

IStdOutFunction

```
public interface IStdOutFunction {
    /**
     * Called when something should be written to the standard
     * output stream.
     *
     * @param callerHandle The caller handle.
     * @param str The string represented by a byte array to write.
     * @param len The number of bytes to write.
     * @return The number of bytes written, must be <code>len</code>.
     */
    public int onStdOut(long callerHandle,
                        byte[] str,
                           int len);
}
```

IStdErrFunction

```
public interface IStdErrFunction {
    /**
     * Called when something should be written to the standard error stream.
     *
     * @param callerHandle The caller handle.
     * @param str The string represented by a byte array to write.
     * @param len The length of bytes to be written.
     * @return The amount of bytes written, must be <code>len</code>.
```

```
     */
    public int onStdErr(long callerHandle,
                        byte[] str,
                           int len);
}
```

IPollFunction

```
public interface IPollFunction {
    public int onPoll(long callerHandle);
}
```

ICalloutFunction

```
public interface ICalloutFunction {
    public int onCallout(long instance,
                         long calloutHandle,
                        byte[] deviceName,
                           int id,
                           int size,
                         long data);
}
```

GSInstance

This is a utility class which makes Ghostscript calls easier by storing a Ghostscript instance and, optionally, a caller handle. Essentially the class acts as a handy wrapper for the standard GSAPI methods.

Constructors

```
public GSInstance() throws IllegalStateException;
```

```
public GSInstance(long callerHandle) throws IllegalStateException;
```

delete_instance

Wraps gsapi_delete_instance.

```
public void delete_instance();
```

set_stdio

Wraps gsapi_set_stdio.

```
public int set_stdio(IStdInFunction stdin,
                     IStdOutFunction stdout,
                     IStdErrFunction stderr);
```

set_poll

Wraps gsapi_set_poll.

```
public int set_poll(IPollFunction pollfun);
```

set_display_callback

Wraps gsapi_set_display_callback.

```
public int set_display_callback(DisplayCallback displaycallback);
```

register_callout

Wraps gsapi_register_callout.

```
public int register_callout(ICalloutFunction callout);
```

deregister_callout

Wraps gsapi_deregister_callout.

```
public void deregister_callout(ICalloutFunction callout);
```

set_arg_encoding

Wraps gsapi_set_arg_encoding.

```
public int set_arg_encoding(int encoding);
```

set_default_device_list

Wraps gsapi_set_default_device_list.

```
public int set_default_device_list(byte[] list,
                                   int listlen);
```

get_default_device_list

Wraps gsapi_get_default_device_list.

```
public int get_default_device_list(Reference<byte[]> list,
                                   Reference<Integer> listlen);
```

init_with_args

Wraps gsapi_init_with_args.

```
public int init_with_args(int argc,
                    byte[][] argv);
```

```
public int init_with_args(String[] argv);
```

```
public int init_with_args(List<String> argv);
```

run_string_begin

Wraps gsapi_run_string_begin.

```
public int run_string_begin(int userErrors,
            Reference<Integer> pExitCode);
```

run_string_continue

Wraps gsapi_run_string_continue.

```
public int run_string_continue(byte[] str,
                               int length,
                               int userErrors,
                  Reference<Integer> pExitCode);
```

```
public int run_string_continue(String str,
                               int length,
                               int userErrors,
                  Reference<Integer> pExitCode);
```

run_string

Wraps gsapi_run_string.

```
public int run_string(byte[] str,
                      int userErrors,
          Reference<Integer> pExitCode);
```

```
public int run_string(String str,
                      int userErrors,
          Reference<Integer> pExitCode);
```

run_file

Wraps gsapi_run_file.

```
public int run_file(byte[] fileName,
                    int userErrors,
        Reference<Integer> pExitCode);
```

```
public int run_file(String filename,
                    int userErrors,
        Reference<Integer> pExitCode);
```

exit

Wraps gsapi_exit.

```
public int exit();
```

set_param

Wraps gsapi_set_param.

```
public int set_param(byte[] param,
                     Object value,
                        int paramType);
```

```
public int set_param(String param,
                     String value,
                        int paramType);
```

```
public int set_param(String param,
                     Object value,
                        int paramType);
```

get_param

Wraps gsapi_get_param.

```
public int get_param(byte[] param,
                      long value,
                       int paramType);
```

```
public int get_param(String param,
                      long value,
                       int paramType);
```

enumerate_params

Wraps gsapi_enumerate_params.

```
public int enumerate_params(Reference<Long> iter,
                            Reference<byte[]> key,
                           Reference<Integer> paramType);
```

add_control_path

Wraps gsapi_add_control_path.

```
public int add_control_path(int type,
                            byte[] path);
```

```
public int add_control_path(int type,
                            String path);
```

remove_control_path

Wraps gsapi_remove_control_path.

```
public int remove_control_path(int type,
                               byte[] path);
```

```
public int remove_control_path(int type,
                               String path);
```

purge_control_paths

Wraps gsapi_purge_control_paths.

```
public void purge_control_paths(int type);
```

activate_path_control

Wraps gsapi_activate_path_control.

```
public void activate_path_control(boolean enable);
```

is_path_control_active

Wraps gsapi_is_path_control_active.

```
public boolean is_path_control_active();
```

Utility classes

The com.artifex.gsjava.util package contains a set of classes and interfaces which are used throughout the API.

com.artifex.gsjava.util.Reference

`Reference<T>` is used in many of the Ghostscript calls, it stores a reference to a generic value of type `T`. This class exists to emulate pointers being passed to a native function. Its value can be fetched with `getValue()` and set with `setValue(T value)`.

```
public class Reference<T> {

    private volatile T value;

    public Reference() {
        this(null);
    }

    public Reference(T value) {
        this.value = value;
    }

    public void setValue(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
    ...
}
```

Python overview

About

The `Python` API is provided by the file gsapi.py - this is the binding to the Ghostscript `C` library.

In the GhostPDL repository sample `Python` examples can be found in `/demos/python/examples.py`.

Platform & setup

Building Ghostscript

Ghostscript should be built as a shared library for your platform.

See Building Ghostscript.

Specifying the Ghostscript shared library

Two environmental variables can be used to specify where to find the Ghostscript shared library.

`GSAPI_LIB` sets the exact path of the Ghostscript shared library, otherwise, `GSAPI_LIBDIR` sets the directory containing the Ghostscript shared library.

If neither is defined we will use the OS's default location(s) for shared libraries.

If `GSAPI_LIB` is not defined, the leafname of the shared library is inferred
from the OS type - `libgs.so` on Unix, `libgs.dylib` on MacOS, `gsdll64.dll` on Windows 64.

API test

The `gsapi.py` file that provides the `Python` bindings can also be used to test the bindings, by running it directly.

Assuming that your Ghostscript library has successfully been created, then from the root of your `ghostpdl` checkout run:

Windows

from ghostpdl

```
// Run gsapi.py as a test script in a cmd.exe window:
set GSAPI_LIBDIR=debugbin&& python ./demos/python/gsapi.py

// Run gsapi.py as a test script in a PowerShell window:
cmd /C "set GSAPI_LIBDIR=debugbin&& python ./demos/python/gsapi.py"
```

Linux/OpenBSD/MacOS

from ghostpdl

```
// Run gsapi.py as a test script:
GSAPI_LIBDIR=sodebugbin ./demos/python/gsapi.py
```

If there are no errors then this will have validated that the Ghostscript library is present & operational.

gsapi.py

About

`gsapi.py` is the `Python` binding into the Ghostscript `C` library.

Assuming that the Ghostscript library has been built for your project then `gsapi` should be imported into your own `Python` scripts for API usage.

```
import gsapi
```

gsapi

Overview

Implemented using Python's ctypes module.

All functions have the same name as the `C` function that they wrap.

Functions raise a `GSError` exception if the underlying function returned a negative error code.

Functions that don't have out-params return `None`. Out-params are returned directly (using tuples if there are more than one).

Return codes

`gsapi_run_*` and `gsapi_exit` methods return an `int` code which can be interpreted as follows:

code	status
`0`	no error
`gsConstants.E_QUIT`	"quit" has been executed. This is not an error. gsapi_exit must be called next
`<0`	error

> **NOTE**
>
> For full detail on these return code please see:
> https://www.ghostscript.com/doc/current/API.htm#return_codes

gsapi_revision

Returns a `gsapi_revision_t`.

This method returns the revision numbers and strings of the Ghostscript interpreter library; you should call it before any other interpreter library functions to make sure that the correct version of the Ghostscript interpreter has been loaded.

```
def gsapi_revision()
```

```
version_info = gsapi.gsapi_revision()
print (version_info)
```

gsapi_new_instance

Returns a new instance of Ghostscript to be used with other `gsapi_*()` functions.

```
def gsapi_new_instance(caller_handle)
```

Parameters

`caller_handle` : Typically unused, but is passed to callbacks e.g. via `gsapi_set_stdio()` . Must be convertible to a `C void*` , so `None` or an
integer is ok but other types such as strings will fail.

```
instance = gsapi.gsapi_new_instance(1)
```

gsapi_delete_instance

Destroy an instance of Ghostscript. Before you call this, Ghostscript should ensure to have finished any processes.

```
def gsapi_delete_instance(instance)
```

Parameters

`instance` : Your instance of Ghostscript.

```
gsapi.gsapi_delete_instance(instance)
```

gsapi_set_stdio

Set the callback functions for `stdio` , together with the handle to use in the callback functions.

```
def gsapi_set_stdio(instance, stdin_fn, stdout_fn, stderr_fn)
```

Parameters

`instance` : Your instance of Ghostscript.

`stdin_fn` : If not `None` , will be called with:

- `(caller_handle, text, len_)` :
  - `caller_handle` : As passed originally to `gsapi_new_instance()` .
  - `text` : A `ctypes.LP_c_char` of length `len_` .

`stdout_fn` and `stderr_fn` :If not `None` , called with:

- `(caller_handle, text)`:
  - `caller_handle`: As passed originally to `gsapi_new_instance()`.
  - `text`: A Python bytes object.

Should return the number of bytes of `text` that they handled; for convenience `None` is converted to `len(text)`.

```
def stdout_fn(caller_handle, bytes_):
    sys.stdout.write(bytes_.decode('latin-1'))

gsapi.gsapi_set_stdio(instance, None, stdout_fn, None)
print('gsapi_set_stdio() ok.')
```

gsapi_set_poll

Set the callback function for polling.

```
def gsapi_set_poll(instance, poll_fn)
```

Parameters

`instance`: Your instance of Ghostscript.

`poll_fn`: Will be called with `caller_handle` as passed to `gsapi_new_instance()`.

```
def poll_fn(caller_handle, bytes_):
    sys.stdout.write(bytes_.decode('latin-1'))

gsapi.gsapi_set_poll(instance, poll_fn)
print('gsapi_set_poll() ok.')
```

gsapi_set_display_callback

Sets the display callback.

```
def gsapi_set_display_callback(instance, callback)
```

Parameters

`instance`: Your instance of Ghostscript.

`callback`: Must be a `display_callback` instance.

```
d = display_callback()
gsapi.gsapi_set_display_callback(instance, d)
print('gsapi_set_display_callback() ok.')
```

gsapi_set_arg_encoding

Set the encoding used for the interpretation of all subsequent arguments supplied via the `GhostAPI` interface on this instance. By default we expect args to be in encoding `0` (the 'local' encoding for this OS). On Windows this means "the currently

selected codepage". On Linux this typically means `utf8`. This means that omitting to call this function will leave Ghostscript running exactly as it always has.

This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
def gsapi_set_arg_encoding(instance, encoding)
```

Parameters

`instance`: Your instance of Ghostscript.

`encoding`: Encoding must be one of:

Encoding enum	Value
`GS_ARG_ENCODING_LOCAL`	0
`GS_ARG_ENCODING_UTF8`	1
`GS_ARG_ENCODING_UTF16LE`	2

```
gsapi.gsapi_set_arg_encoding(instance, gsapi.GS_ARG_ENCODING_UTF8)
```

> **NOTE**
>
> Please note that use of the 'local' encoding (`GS_ARG_ENCODING_LOCAL`) is now deprecated and should be avoided in new code.

gsapi_set_default_device_list

Set the string containing the list of default device names, for example "display x11alpha x11 bbox". Allows the calling application to influence which device(s) Ghostscript will try, in order, in its selection of the default device. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
def gsapi_set_default_device_list(instance, list_)
```

Parameters

`instance`: Your instance of Ghostscript.

`list_`: A string of device names.

```
gsapi.gsapi_set_default_device_list(instance, 'bmp256 bmp32b bmpgray cdeskjet cdj1600 cdj500')
```

gsapi_get_default_device_list

Returns a string containing the list of default device names. This must be called after gsapi_new_instance and before gsapi_init_with_args.

```
def gsapi_get_default_device_list(instance)
```

Parameters

`instance` : Your instance of Ghostscript.

```
device_list = gsapi.gsapi_get_default_device_list(instance)
print(device_list)
```

gsapi_init_with_args

To initialise the interpreter, pass your `instance` of Ghostscript and your argument variables with `args`.

```
def gsapi_init_with_args(instance, args)
```

Parameters

`instance` : Your instance of Ghostscript.

`args` : A list/tuple of strings.

```
in_filename = 'tiger.eps'
out_filename = 'tiger.pdf'
params = ['gs', '-dNOPAUSE', '-dBATCH', '-sDEVICE=pdfwrite',
          '-o', out_filename, '-f', in_filename]
gsapi.gsapi_init_with_args(instance, params)
```

gsapi_run_*

Returns an exit code or an exception on error.

There is a 64 KB length limit on any buffer submitted to a `gsapi_run_*` function for processing. If you have more than 65535 bytes of input then you must split it into smaller pieces and submit each in a separate gsapi_run_string_continue call.

> **NOTE**
>
> All these functions return an exit code

gsapi_run_string_begin

Starts a `run_string_` operation.

```
def gsapi_run_string_begin(instance, user_errors)
```

Parameters

`instance` : Your instance of Ghostscript.

`user_errors` : An `int` , for more see user errors parameter explained.

```
exitcode = gsapi.gsapi_run_string_begin(instance, 0)
```

gsapi_run_string_continue

Processes file byte data (`str_`) to feed as chunks into Ghostscript. This method should typically be called within a buffer context.

> **NOTE**
>
> An exception is *not* raised for the `gs_error_NeedInput` return code.

```
def gsapi_run_string_continue(instance, str_, user_errors)
```

Parameters

`instance` : Your instance of Ghostscript.

`str_` : Should be either a Python string or a bytes object. If the former, it is converted into a bytes object using `utf-8` encoding.

`user_errors` : An `int` , for more see user errors parameter explained.

```
exitcode = gsapi.gsapi_run_string_continue(instance, data, 0)
```

> **NOTE**
>
> For the return code, we don't raise an exception for `gs_error_NeedInput` .

gsapi_run_string_with_length

Processes file byte data (`str_`) to feed into Ghostscript when the `length` is known and the file byte data is immediately available.

```
def gsapi_run_string_with_length(instance, str_, length, user_errors)
```

Parameters

`instance` : Your instance of Ghostscript.

`str_` : Should be either a Python string or a bytes object. If the former, it is converted into a bytes object using `utf-8` encoding.

`length` : An `int` representing the length of `gsapi_run_string_with_length` .

`user_errors` : An `int` , for more see user errors parameter explained.

```
gsapi.gsapi_run_string_with_length(instance,"hello",5,0)
```

> **NOTE**
>
> If using this method then ensure that the file byte data will fit into a single (<64k) buffer.

gsapi_run_string

Processes file byte data (`str_`) to feed into Ghostscript.

```
def gsapi_run_string(instance, str_, user_errors)
```

Parameters

`instance` : Your instance of Ghostscript.

`str_` : Should be either a Python string or a bytes object. If the former, it is converted into a bytes object using `utf-8` encoding.

`user_errors` : An `int` , for more see user errors parameter explained.

```
gsapi.gsapi_run_string(instance,"hello",0)
```

> **NOTE**
>
> This method can only work on a standard, null terminated C string.

gsapi_run_string_end

Ends a `run_string_` operation.

```
def gsapi_run_string_end(instance, user_errors)
```

Parameters

`instance` : Your instance of Ghostscript.

`user_errors` : An `int` , for more see user errors parameter explained.

```
exitcode = gsapi.gsapi_run_string_end(instance, 0)
```

gsapi_run_file

Runs a file through Ghostscript.

```
def gsapi_run_file(instance, filename, user_errors)
```

Parameters

`instance` : Your instance of Ghostscript.

`filename` : String representing file name.

`user_errors` : An `int` , for more see user errors parameter explained.

```
in_filename = 'tiger.eps'
gsapi.gsapi_run_file(instance, in_filename, 0)
```

> **NOTE**
>
> This will process the supplied input file with any previously supplied argument parameters.

gsapi_exit

Exit the interpreter. This must be called on shutdown if gsapi_init_with_args has been called, and just before gsapi_delete_instance.

```
def gsapi_exit(instance)
```

```
gsapi.gsapi_exit(instance)
```

gsapi_set_param

Sets a parameter. Broadly, this is equivalent to setting a parameter using -d, -s or -p on the command line. This call cannot be made during a gsapi_run_string operation.

Parameters in this context are not the same as 'arguments' as processed by gsapi_init_with_args, but often the same thing can be achieved. For example, with gsapi_init_with_args, we can pass "-r200" to change the resolution. Broadly the same thing can be achieved by using gsapi_set_param to set a parsed value of "<>".

Internally, when we set a parameter, we perform an `initgraphics` operation. This means that using gsapi_set_param other than at the start of a page is likely to give unexpected results.

Attempting to set a parameter that the device does not recognise will be silently ignored, and that parameter will not be found in subsequent gsapi_get_param calls.

```
def gsapi_set_param(instance, param, value, type_=None)
```

Parameters

`instance` : Your instance of Ghostscript.

`param` : Name of parameter, either a bytes or a str; if str it is encoded using latin-1.

`value` : A bool, int, float, bytes or str. If str, it is encoded into a bytes using utf-8.

`type_` : If `type_` is not `None` , `value` must be convertible to the Python type implied by `type_` :

Parameter list

`type_`	Python type(s)

`type_`	Python type(s)
gs_spt_null	[Ignored]
gs_spt_bool	bool
gs_spt_int	int
gs_spt_float	float
gs_spt_name	[Error]
gs_spt_string	(bytes, str)
gs_spt_long	int
gs_spt_i64	int
gs_spt_size_t	int
gs_spt_parsed	(bytes, str)
gs_spt_more_to_come	(bytes, str)

An exception is raised if `type_` is an integer type and `value` is outside its range.

If `type_` is `None`, we choose something suitable for type of `value`:

Python type of `value`	`type_`
bool	gs_spt_bool
int	gs_spt_i64
float	gs_spt_float
bytes	gs_spt_parsed
str	gs_spt_parsed (encoded with utf-8)

If `value` is `None`, we use `gs_spt_null`.

Otherwise `type_` must be a `gs_spt_*` except for `gs_spt_invalid` and `gs_spt_name`.

> **NOTE**
>
> This implementation supports automatic inference of type by looking at the type of `value`.

```
set_margins = gsapi.gsapi_set_param(instance, "Margins", "[10 10]")
```

> **NOTE**
>
> For more on the `C` implementation of parameters see: Ghostscript parameters in C.

gsapi_get_param

Retrieve the current value of a parameter.

If an error occurs, the return value is negative. Otherwise the return value is the number of bytes required for storage of the value. Call once with value `NULL` to get the number of bytes required, then call again with value pointing to at least the required number of bytes where the value will be copied out. Note that the caller is required to know the type of value in order to get it. For all types other than gs_spt_string, gs_spt_name, and gs_spt_parsed knowing the type means you already know the size required.

This call retrieves parameters/values that have made it to the device. Thus, any values set using gs_spt_more_to_come without a following call omitting that flag will not be retrieved. Similarly, attempting to get a parameter before gsapi_init_with_args has been called will not list any, even if gsapi_set_param has been used.

Attempting to read a parameter that is not set will return `gs_error_undefined` (-21). Note that calling gsapi_set_param followed by gsapi_get_param may not find the value, if the device did not recognise the key as being one of its configuration keys.

For the `C` documentation please refer to Ghostscript get_param.

```
def gsapi_get_param(instance, param, type_=None, encoding=None)
```

Parameters

`instance` : Your instance of Ghostscript.

`param` : Name of parameter, either a `bytes` or `str` ; if a `str` it is encoded using `latin-1` .

`type_` : A `gs_spt_*` constant or `None` . If `None` we try each `gs_spt_*` until one succeeds; if none succeeds we raise the last error.

`encoding` : Only affects string values. If `None` we return a `bytes` object, otherwise it should be the encoding to use to decode into a string, e.g. 'utf-8'.

```
get_margins = gsapi.gsapi_get_param(instance, "Margins")
```

gsapi_enumerate_params

Enumerate the current parameters on the instance of Ghostscript.

Returns an array of `(key, value)` for each parameter. `key` is decoded as `latin-1` .

```
def gsapi_enumerate_params(instance)
```

Parameters

`instance` : Your instance of Ghostscript.

```
for param, type_ in gsapi.gsapi_enumerate_params(instance):
    val = gsapi.gsapi_get_param(instance,param, encoding='utf-8')
    print('%-24s : %s' % (param, val))
```

gsapi_add_control_path

Add a (case sensitive) path to one of the lists of permitted paths for file access.

```
def gsapi_add_control_path(instance, type_, path)
```

Parameters

`instance` : Your instance of Ghostscript.

`type_` : An `int` which must be one of:

Enum	Value
GS_PERMIT_FILE_READING	0
GS_PERMIT_FILE_WRITING	1
GS_PERMIT_FILE_CONTROL	2

`path` : A `string` representing the file path.

```
gsapi.gsapi_add_control_path(instance, gsapi.GS_PERMIT_FILE_READING, "/docs/secure/")
```

gsapi_remove_control_path

Remove a (case sensitive) path from one of the lists of permitted paths for file access.

```
def gsapi_remove_control_path(instance, type_, path)
```

Parameters

`instance` : Your instance of Ghostscript.

`type_` : An `int` representing the permission type.

`path` : A `string` representing the file path.

```
gsapi.gsapi_remove_control_path(instance, gsapi.GS_PERMIT_FILE_READING, "/docs/secure/")
```

gsapi_purge_control_paths

Clear all the paths from one of the lists of permitted paths for file access.

```
def gsapi_purge_control_paths(instance, type_)
```

Parameters

`instance` : Your instance of Ghostscript.

`type_` : An `int` representing the permission type.

```
gsapi.gsapi_purge_control_paths(instance, gsapi.GS_PERMIT_FILE_READING)
```

gsapi_activate_path_control

Enable/Disable path control (i.e. whether paths are checked against permitted paths before access is granted).

```
def gsapi_activate_path_control(instance, enable)
```

Parameters

`instance` : Your instance of Ghostscript.

`enable` : `bool` to enable/disable path control.

```
gsapi.gsapi_activate_path_control(instance, true)
```

gsapi_is_path_control_active

Query whether path control is activated or not.

```
def gsapi_is_path_control_active(instance)
```

Parameters

`instance` : Your instance of Ghostscript.

```
isActive = gsapi.gsapi_is_path_control_active(instance)
```

Notes

1: User errors parameter

The `user_errors` argument is normally set to zero to indicate that errors should be handled through the normal mechanisms within the interpreted code. If set to a negative value, the functions will return an error code directly to the caller, bypassing the interpreted language. The interpreted language's error handler is bypassed, regardless of `user_errors` parameter, for the `gs_error_interrupt` generated when the polling callback returns a negative value. A positive `user_errors` is treated the same as zero.