

---

# **MuPDF Documentation**

***Release 1.21.2***

**Artifex**

**Jun 01, 2023**



# USER GUIDE

<b>1</b>	<b>Quick Start Guide</b>	<b>3</b>
1.1	Get the <i>MuPDF</i> source code . . . . .	3
1.2	Building the library . . . . .	3
1.3	Validating your installation . . . . .	4
1.4	Supported file formats . . . . .	4
<b>2</b>	<b>Using <i>MuPDF</i> with C</b>	<b>5</b>
2.1	Basic <i>MuPDF</i> usage example . . . . .	5
2.2	Common function arguments . . . . .	8
2.3	Error handling . . . . .	8
2.4	Multi-threading . . . . .	10
2.5	Cloning the context . . . . .	18
2.6	Coding Style . . . . .	18
<b>3</b>	<b><i>MuPDF</i> on the command line</b>	<b>21</b>
3.1	<i>mupdf-gl</i> . . . . .	21
3.2	<i>muraster</i> . . . . .	24
3.3	<i>mutool</i> . . . . .	25
3.4	<i>mutool</i> JavaScript API . . . . .	51
<b>4</b>	<b>C API</b>	<b>121</b>
4.1	<i>MuPDF</i> modules . . . . .	121
<b>5</b>	<b>Language Bindings</b>	<b>141</b>
5.1	The C++ <i>MuPDF</i> API . . . . .	141
5.2	The Python and C# <i>MuPDF</i> APIs . . . . .	144
5.3	Installing the Python <i>mupdf</i> module using <i>pip</i> . . . . .	145
5.4	Doxygen/Pydoc API documentation . . . . .	145
5.5	Example client code . . . . .	145
5.6	Changelog . . . . .	147
5.7	Building the C++, Python and C# <i>MuPDF</i> APIs from source . . . . .	151
5.8	Windows-specifics . . . . .	156
5.9	C++ bindings details . . . . .	157
5.10	Python/C# bindings details . . . . .	159
<b>6</b>	<b>Progressive Loading</b>	<b>163</b>
6.1	What is progressive loading? . . . . .	163
6.2	Progressive download using “linearized” files . . . . .	163
6.3	The Hint Stream . . . . .	164
6.4	So how does <i>MuPDF</i> handle progressive loading? . . . . .	164
6.5	Progressive loading using byte range requests . . . . .	165

<b>7</b>	<b>Changes</b>	<b>167</b>
7.1	Changes to the existing <i>API</i> . . . . .	167
<b>8</b>	<b>Third Party Libraries Used by <i>MuPDF</i></b>	<b>171</b>



*MuPDF* is a lightweight *PDF*, *XPS*, and *E-book* viewer.

*MuPDF* consists of a software library, command line tools, and viewers for various platforms.

To get started with *MuPDF* a developer should *acquire the source code* and *build the library*.



## QUICK START GUIDE

### 1.1 Get the *MuPDF* source code

There are a few options for acquiring the source code as follows:

- Download an official *MuPDF* release here: [MuPDF Releases](#).
- Check out the *Github* repository here: [MuPDF on Github](#).
- Or check out the latest development source directly from our canonical git repository: `git clone --recursive git://git.ghostscript.com/mupdf.git`.

---

**Note:** If you have checked out a *git repo* then in the “mupdf” directory, update the third party libraries with: `git submodule update --init`.

---

### 1.2 Building the library

#### 1.2.1 *Windows*

On *Windows* there is a *Visual Studio* solution file in `platform/win32/mupdf.sln`.

##### Using Microsoft Visual Studio

To build the required DLLs, load `platform/win32/mupdf.sln` into *Visual Studio*, and select the required architecture from the drop down - then right click on `libmupdf` in the solution explorer and choose “Build”.

#### 1.2.2 *macOS*

Build for *macOS* with the following *Terminal* command:

```
make prefix=/usr/local install
```

This will then install the viewer, command line tools, libraries, and header files on your system.

### 1.2.3 Linux

You will also need the X11 headers and libraries if you're building on *Linux*. These can typically be found in the *xorg-dev* package.

Alternatively, if you only want the command line tools, you can build with `HAVE_X11=no`.

The new *OpenGL* based viewer also needs *OpenGL* headers and libraries. If you're building on *Linux*, install the *mesa-common-dev*, *libgl1-mesa-dev* packages, and *libglu1-mesa-dev* packages. You'll also need several X11 development packages: *xorg-dev*, *libxcursor-dev*, *libxrandr-dev*, and *libxinerama-dev*. To skip building the *OpenGL* viewer, build with `HAVE_GLUT=no`.

To install the viewer, command line tools, libraries, and header files on your system:

```
make prefix=/usr/local install
```

To install only the command line tools, libraries, and headers invoke `make` like this:

```
make HAVE_X11=no HAVE_GLUT=no prefix=/usr/local install
```

---

**Note:** Results of the build can be found in `build/release` in your “mupdf” folder.

Building for both *macOS* & *Linux* depend on having both `autoconf` and `automake` installed on your system. If this software is not already on your system (usually `/usr/local/bin`) then it can be installed from the [GNU website](#).

Alternatively, `autoconf` and `automake` can be installed via *Brew* by running: `brew install autoconf automake`.

---

## 1.3 Validating your installation

From the command line you should now have the following *MuPDF* commands available:

- *mupdf-gl* A handy viewer UI.
- *muraster* Can be used to convert PDF pages to raster images.
- *mutool* An all purpose tool for dealing with PDF files.

## 1.4 Supported file formats

*MuPDF* supports the following file formats:

pdf, epub, xps, cbz, mobi, fb2, svg

And a suite of image types, e.g. png, jpg, bmp etc.



## USING *MUPDF* WITH C

This intends to be an introductory guide into how to use *MuPDF* with your C code and as such assumes at least an intermediate knowledge of programming with C.

### 2.1 Basic *MuPDF* usage example

For an example of how to use *MuPDF* in the most basic way, see `docs/examples/example.c`.

Listing 1: `docs/examples/example.c`

```
/*
How to use MuPDF to render a single page and print the result as
a PPM to stdout.

To build this example in a source tree and render first page at
100% zoom with no rotation, run:
make examples
./build/debug/example document.pdf 1 100 0 > page1.ppm

To build from installed sources, and render the same document, run:
gcc -I/usr/local/include -o example \
    /usr/local/share/doc/mupdf/examples/example.c \
    /usr/local/lib/libmupdf.a \
    /usr/local/lib/libmupdfthird.a \
    -lm
./example document.pdf 1 100 0 > page1.ppm
*/

#include <mupdf/fitz.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    char *input;
    float zoom, rotate;
    int page_number, page_count;
    fz_context *ctx;
    fz_document *doc;
```

(continues on next page)

(continued from previous page)

```

    fz_pixmap *pix;
    fz_matrix ctm;
    int x, y;

    if (argc < 3)
    {
        fprintf(stderr, "usage: example input-file page-number [ zoom [ rotate ] ]\n");
        fprintf(stderr, "\tinput-file: path of PDF, XPS, CBZ or EPUB document to open\n");
        fprintf(stderr, "\tPage numbering starts from one.\n");
        fprintf(stderr, "\tZoom level is in percent (100 percent is 72 dpi).\n");
        fprintf(stderr, "\tRotation is in degrees clockwise.\n");
        return EXIT_FAILURE;
    }

    input = argv[1];
    page_number = atoi(argv[2]) - 1;
    zoom = argc > 3 ? atof(argv[3]) : 100;
    rotate = argc > 4 ? atof(argv[4]) : 0;

    /* Create a context to hold the exception stack and various caches. */
    ctx = fz_new_context(NULL, NULL, FZ_STORE_UNLIMITED);
    if (!ctx)
    {
        fprintf(stderr, "cannot create mupdf context\n");
        return EXIT_FAILURE;
    }

    /* Register the default file types to handle. */
    fz_try(ctx)
        fz_register_document_handlers(ctx);
    fz_catch(ctx)
    {
        fprintf(stderr, "cannot register document handlers: %s\n", fz_caught_message(ctx));
        fz_drop_context(ctx);
        return EXIT_FAILURE;
    }

    /* Open the document. */
    fz_try(ctx)
        doc = fz_open_document(ctx, input);
    fz_catch(ctx)
    {
        fprintf(stderr, "cannot open document: %s\n", fz_caught_message(ctx));
        fz_drop_context(ctx);
        return EXIT_FAILURE;
    }

    /* Count the number of pages. */
    fz_try(ctx)

```

(continues on next page)

(continued from previous page)

```

        page_count = fz_count_pages(ctx, doc);
        fz_catch(ctx)
        {
            fprintf(stderr, "cannot count number of pages: %s\n", fz_caught_
↪message(ctx));
            fz_drop_document(ctx, doc);
            fz_drop_context(ctx);
            return EXIT_FAILURE;
        }

        if (page_number < 0 || page_number >= page_count)
        {
            fprintf(stderr, "page number out of range: %d (page count %d)\n", page_
↪number + 1, page_count);
            fz_drop_document(ctx, doc);
            fz_drop_context(ctx);
            return EXIT_FAILURE;
        }

        /* Compute a transformation matrix for the zoom and rotation desired. */
        /* The default resolution without scaling is 72 dpi. */
        ctm = fz_scale(zoom / 100, zoom / 100);
        ctm = fz_pre_rotate(ctm, rotate);

        /* Render page to an RGB pixmap. */
        fz_try(ctx)
        {
            pix = fz_new_pixmap_from_page_number(ctx, doc, page_number, ctm, fz_
↪device_rgb(ctx), 0);
            fz_catch(ctx)
            {
                fprintf(stderr, "cannot render page: %s\n", fz_caught_message(ctx));
                fz_drop_document(ctx, doc);
                fz_drop_context(ctx);
                return EXIT_FAILURE;
            }

            /* Print image data in ascii PPM format. */
            printf("P3\n");
            printf("%d %d\n", pix->w, pix->h);
            printf("255\n");
            for (y = 0; y < pix->h; ++y)
            {
                unsigned char *p = &pix->samples[y * pix->stride];
                for (x = 0; x < pix->w; ++x)
                {
                    if (x > 0)
                        printf(" ");
                    printf("%3d %3d %3d", p[0], p[1], p[2]);
                    p += pix->n;
                }
                printf("\n");
            }
        }

```

(continues on next page)

(continued from previous page)

```
/* Clean up. */
fz_drop_pixmap(ctx, pix);
fz_drop_document(ctx, doc);
fz_drop_context(ctx);
return EXIT_SUCCESS;
}
```

To limit the complexity and give an easier introduction this code has no error handling at all, but any serious piece of code using *MuPDF* should use *error handling strategies*.

## 2.2 Common function arguments

Most functions in *MuPDF*'s interface take a context argument.

A context contains global state used by *MuPDF* inside functions when parsing or rendering pages of the document. It contains for example:

- an exception stack (see error handling below),
- a memory allocator (allowing for custom allocators)
- a resource store (for caching of images, fonts, etc.)
- a set of locks and (un-)locking functions (for multi-threading)

Without the set of locks and accompanying functions the context and its proxies may only be used in a single-threaded application.

## 2.3 Error handling

*MuPDF* uses a set of exception handling macros to simplify error return and cleanup. Conceptually, they work a lot like C++'s try/catch system, but do not require any special compiler support.

The basic formulation is as follows:

```
fz_try(ctx)
{
    // Try to perform a task. Never 'return', 'goto' or
    // 'longjmp' out of here. 'break' may be used to
    // safely exit (just) the try block scope.
}
fz_always(ctx)
{
    // Any code here is always executed, regardless of
    // whether an exception was thrown within the try or
    // not. Never 'return', 'goto' or longjmp out from
    // here. 'break' may be used to safely exit (just) the
    // always block scope.
}
fz_catch(ctx)
{
}
```

(continues on next page)

(continued from previous page)

```

// This code is called (after any always block) only
// if something within the fz_try block (including any
// functions it called) threw an exception. The code
// here is expected to handle the exception (maybe
// record/report the error, cleanup any stray state
// etc) and can then either exit the block, or pass on
// the exception to a higher level (enclosing) fz_try
// block (using fz_throw, or fz_rethrow).
}

```

The `fz_always` block is optional, and can safely be omitted.

The macro based nature of this system has 3 main limitations:

Never return from within try (or ‘goto’ or `longjmp` out of it). This upsets the internal housekeeping of the macros and will cause problems later on. The code will detect such things happening, but by then it is too late to give a helpful error report as to where the original infraction occurred.

The `fz_try(ctx) { ... } fz_always(ctx) { ... } fz_catch(ctx) { ... }` is not one atomic *C* statement. That is to say, if you do:

```

if (condition)
    fz_try(ctx) { ... }
    fz_catch(ctx) { ... }

```

then you will not get what you want. Use the following instead:

```

if (condition) {
    fz_try(ctx) { ... }
    fz_catch(ctx) { ... }
}

```

The macros are implemented using `setjmp` and `longjmp`, and so the standard *C* restrictions on the use of those functions apply to `fz_try` / `fz_catch` too. In particular, any “truly local” variable that is set between the start of `fz_try` and something in `fz_try` throwing an exception may become undefined as part of the process of throwing that exception.

As a way of mitigating this problem, we provide a `fz_var()` macro that tells the compiler to ensure that that variable is not unset by the act of throwing the exception.

A model piece of code using these macros then might be:

```

house build_house(plans *p)
{
    material m = NULL;
    walls w = NULL;
    roof r = NULL;
    house h = NULL;
    tiles t = make_tiles();

    fz_var(w);
    fz_var(r);
    fz_var(h);

    fz_try(ctx)

```

(continues on next page)

(continued from previous page)

```

{
    fz_try(ctx)
    {
        m = make_bricks();
    }
    fz_catch(ctx)
    {
        // No bricks available, make do with straw?
        m = make_straw();
    }
    w = make_walls(m, p);
    r = make_roof(m, t);
    // Note, NOT: return combine(w,r);
    h = combine(w, r);
}
fz_always(ctx)
{
    drop_walls(w);
    drop_roof(r);
    drop_material(m);
    drop_tiles(t);
}
fz_catch(ctx)
{
    fz_throw(ctx, "build_house failed");
}
return h;
}

```

Things to note about this:

If `make_tiles` throws an exception, this will immediately be handled by some higher level exception handler. If it succeeds, `t` will be set before `fz_try` starts, so there is no need to `fz_var(t);`.

We try first off to make some bricks as our building material. If this fails, we fall back to straw. If this fails, we'll end up in the `fz_catch`, and the process will fail neatly.

We assume in this code that `combine` takes new reference to both the walls and the roof it uses, and therefore that `w` and `r` need to be cleaned up in all cases.

We assume the standard *C* convention that it is safe to destroy `NULL` things.

## 2.4 Multi-threading

First off, study the basic usage example in [docs/examples/example.c](#) and make sure you understand how it works as the data structures manipulated there will be referred to in this section too.

*MuPDF* can usefully be built into a multi-threaded application without the library needing to know anything about threading at all. If the library opens a document in one thread, and then sits there as a 'server' requesting pages and rendering them for other threads that need them, then the library is only ever being called from this one thread.

Other threads can still be used to handle UI requests etc, but as far as *MuPDF* is concerned it is only being used in a single threaded way. In this instance, there are no threading issues with *MuPDF* at all, and it can safely be used without any locking, as described in the previous sections.

This section will attempt to explain how to use *MuPDF* in the more complex case; where we genuinely want to call the *MuPDF* library concurrently from multiple threads within a single application.

*MuPDF* can be invoked with a user supplied set of locking functions. It uses these to take mutexes around operations that would conflict if performed concurrently in multiple threads. By leaving the exact implementation of locks to the caller *MuPDF* remains threading library agnostic.

The following simple rules should be followed to ensure that multi-threaded operations run smoothly:

1. **“No simultaneous calls to MuPDF in different threads are allowed to use the same context.”**

Most of the time it is simplest to just use a different context for every thread; just create a new context at the same time as you create the thread. For more details see “Cloning the context” below.

2. **“No simultaneous calls to MuPDF in different threads are allowed to use the same document.”**

Only one thread can be accessing a document at a time, but once display lists are created from that document, multiple threads at a time can operate on them.

The document can be used from several different threads as long as there are safeguards in place to prevent the usages being simultaneous.

3. **“No simultaneous calls to MuPDF in different threads are allowed to use the same device.”**

Calling a device simultaneously from different threads will cause it to get confused and may crash. Calling a device from several different threads is perfectly acceptable as long as there are safeguards in place to prevent the calls being simultaneous.

4. **“An `fz_locks_context` must be supplied at context creation time, unless MuPDF is to be used purely in a single thread at a time.”**

*MuPDF* needs to protect against unsafe access to certain structures/resources/libraries from multiple threads. It does this by using the user supplied locking functions. This holds true even when using completely separate instances of *MuPDF*.

5. **“All contexts in use must share the same `fz_locks_context` (or the underlying locks thereof).”**

We strongly recommend that `fz_new_context` is called just once, and `fz_clone_context` is called to generate new contexts from that. This will automatically ensure that the same locking mechanism is used in all *MuPDF* instances. For now, we do support multiple completely independent contexts being created using repeated calls to `fz_new_context`, but these **MUST** share the same `fz_locks_context` (or at least depend upon the same underlying locks). The facility to create different independent contexts may be removed in future.

### So, how does a multi-threaded example differ from a non-multithreaded one?

Firstly, when we create the first context, we call `fz_new_context` as before, but the second argument should be a pointer to a set of locking functions.

The calling code should provide `FZ_LOCK_MAX` mutexes, which will be locked/unlocked by *MuPDF* calling the lock/unlock function pointers in the supplied structure with the user pointer from the structure and the lock number, `i` ( $0 \leq i < \text{FZ\_LOCK\_MAX}$ ). These mutexes can safely be recursive or non-recursive as *MuPDF* only calls in a non-recursive style.

To make subsequent contexts, the user should **not** call `fz_new_context` again (as this will fail to share important resources such as the store and glyph cache), but should rather call `fz_clone_context`. Each of these cloned contexts can be freed by `fz_free_context` as usual. They will share the important data structures (like store, glyph cache etc.) with the original context, but will have their own exception stacks.

To open a document, call `fz_open_document` as usual, passing a context and a filename. It is important to realise that only one thread at a time can be accessing the documents itself.

This means that only one thread at a time can perform operations such as fetching a page, or rendering that page to a display list. Once a display list has been obtained however, it can be rendered from any other thread (or even from several threads simultaneously, giving banded rendering).

This means that an implementer has 2 basic choices when constructing an application to use *MuPDF* in multi-threaded mode. Either they can construct it so that a single nominated thread opens the document and then acts as a ‘server’ creating display lists for other threads to render, or they can add their own mutex around calls to *MuPDF* that use the document. The former is likely to be far more efficient in the long run.

For an example of how to do multi-threading see below which has a main thread and one rendering thread per page.

Listing 2: docs/examples/multi-threaded.c

```
/*
Multi-threaded rendering of all pages in a document to PNG images.

First look at docs/example.c and make sure you understand it.
Then read the multi-threading section in docs/coding-overview.html.
before coming back here to see an example of multi-threading.

This example will create one main thread for reading pages from the
document, and one thread per page for rendering. After rendering
the main thread will wait for each rendering thread to complete before
writing that thread's rendered image to a PNG image. There is
nothing in MuPDF requiring a rendering thread to only render a
single page, this is just a design decision taken for this example.

To build this example in a source tree and render every page as a
separate PNG, run:
make examples
./build/debug/multi-threaded document.pdf

To build from installed sources, and render the same document, run:
gcc -I/usr/local/include -o multi-threaded \
    /usr/local/share/doc/mupdf/examples/multi-threaded.c \
    /usr/local/lib/libmupdf.a \
    /usr/local/lib/libmupdfthird.a \
    -lpthread -lm
./multi-threaded document.pdf

Caution! As all pages are rendered simultaneously, please choose a
file with just a few pages to avoid stressing your machine too
much. Also you may run in to a limitation on the number of threads
depending on your environment.
*/

//Include the MuPDF header file, and pthread's header file.
#include <mupdf/fitz.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// A convenience function for dying abruptly on pthread errors.
```

(continues on next page)



(continued from previous page)

```

void
fail(const char *msg)
{
    fprintf(stderr, "%s\n", msg);
    abort();
}

// The data structure passed between the requesting main thread and
// each rendering thread.

struct thread_data {
    // A pointer to the original context in the main thread sent
    // from main to rendering thread. It will be used to create
    // each rendering thread's context clone.
    fz_context *ctx;

    // Page number sent from main to rendering thread for printing
    int pagenumber;

    // The display list as obtained by the main thread and sent
    // from main to rendering thread. This contains the drawing
    // commands (text, images, etc.) for the page that should be
    // rendered.
    fz_display_list *list;

    // The area of the page to render as obtained by the main
    // thread and sent from main to rendering thread.
    fz_rect bbox;

    // This is the result, a pixmap containing the rendered page.
    // It is passed first from main thread to the rendering
    // thread, then its samples are changed by the rendering
    // thread, and then back from the rendering thread to the main
    // thread.
    fz_pixmap *pix;

    // This is a note of whether a given thread failed or not.
    int failed;
};

// This is the function run by each rendering function. It takes
// pointer to an instance of the data structure described above and
// renders the display list into the pixmap before exiting.

void *
renderer(void *data_)
{
    struct thread_data *data = (struct thread_data *)data_;
    int pagenumber = data->pagenumber;
    fz_context *ctx = data->ctx;
    fz_display_list *list = data->list;
    fz_rect bbox = data->bbox;

```

(continues on next page)

(continued from previous page)

```

    fz_device *dev = NULL;

    fprintf(stderr, "thread at page %d loading!\n", pagenumber);

    // The context pointer is pointing to the main thread's
    // context, so here we create a new context based on it for
    // use in this thread.
    ctx = fz_clone_context(ctx);

    // Next we run the display list through the draw device which
    // will render the request area of the page to the pixmap.

    fz_var(dev);

    fprintf(stderr, "thread at page %d rendering!\n", pagenumber);
    fz_try(ctx)
    {
        // Create a white pixmap using the correct dimensions.
        data->pix = fz_new_pixmap_with_bbox(ctx, fz_device_rgb(ctx), fz_round_
↪rect(bbox), NULL, 0);
        fz_clear_pixmap_with_value(ctx, data->pix, 0xff);

        // Do the actual rendering.
        dev = fz_new_draw_device(ctx, fz_identity, data->pix);
        fz_run_display_list(ctx, list, dev, fz_identity, bbox, NULL);
        fz_close_device(ctx, dev);
    }
    fz_always(ctx)
        fz_drop_device(ctx, dev);
    fz_catch(ctx)
        data->failed = 1;

    // Free this thread's context.
    fz_drop_context(ctx);

    fprintf(stderr, "thread at page %d done!\n", pagenumber);

    return data;
}

// These are the two locking functions required by MuPDF when
// operating in a multi-threaded environment. They each take a user
// argument that can be used to transfer some state, in this case a
// pointer to the array of mutexes.

void lock_mutex(void *user, int lock)
{
    pthread_mutex_t *mutex = (pthread_mutex_t *) user;

    if (pthread_mutex_lock(&mutex[lock]) != 0)
        fail("pthread_mutex_lock()");
}

```

(continues on next page)

(continued from previous page)

```

void unlock_mutex(void *user, int lock)
{
    pthread_mutex_t *mutex = (pthread_mutex_t *) user;

    if (pthread_mutex_unlock(&mutex[lock]) != 0)
        fail("pthread_mutex_unlock()");
}

int main(int argc, char **argv)
{
    char *filename = argc >= 2 ? argv[1] : "";
    pthread_t *thread = NULL;
    fz_locks_context locks;
    pthread_mutex_t mutex[FZ_LOCK_MAX];
    fz_context *ctx;
    fz_document *doc = NULL;
    int threads;
    int i;

    // Initialize FZ_LOCK_MAX number of non-recursive mutexes.
    for (i = 0; i < FZ_LOCK_MAX; i++)
    {
        if (pthread_mutex_init(&mutex[i], NULL) != 0)
            fail("pthread_mutex_init()");
    }

    // Initialize the locking structure with function pointers to
    // the locking functions and to the user data. In this case
    // the user data is a pointer to the array of mutexes so the
    // locking functions can find the relevant lock to change when
    // they are called. This way we avoid global variables.
    locks.user = mutex;
    locks.lock = lock_mutex;
    locks.unlock = unlock_mutex;

    // This is the main thread's context function, so supply the
    // locking structure. This context will be used to parse all
    // the pages from the document.
    ctx = fz_new_context(NULL, &locks, FZ_STORE_UNLIMITED);

    fz_var(thread);
    fz_var(doc);

    fz_try(ctx)
    {
        // Register default file types.
        fz_register_document_handlers(ctx);

        // Open the PDF, XPS or CBZ document.
        doc = fz_open_document(ctx, filename);
    }
}

```

(continues on next page)

(continued from previous page)

```

// Retrieve the number of pages, which translates to the
// number of threads used for rendering pages.
threads = fz_count_pages(ctx, doc);
fprintf(stderr, "spawning %d threads, one per page...\n", threads);

thread = malloc(threads * sizeof (*thread));

for (i = 0; i < threads; i++)
{
    fz_page *page;
    fz_rect bbox;
    fz_display_list *list;
    fz_device *dev = NULL;
    fz_pixmap *pix;
    struct thread_data *data;

    fz_var(dev);

    fz_try(ctx)
    {
        // Load the relevant page for each thread. Note, that
        // cannot be done on the worker threads, as only one
        // at a time can ever be accessing the document.
        page = fz_load_page(ctx, doc, i);

        // Compute the bounding box for each page.
        bbox = fz_bound_page(ctx, page);

        // Create a display list that will hold the drawing
        // commands for the page. Once we have the display list
        // this can safely be used on any other thread.
        list = fz_new_display_list(ctx, bbox);

        // Create a display list device to populate the page's
        dev = fz_new_list_device(ctx, list);

        // Run the page to that device.
        fz_run_page(ctx, page, dev, fz_identity, NULL);

        // Close the device neatly, so everything is flushed to
        the list.
        fz_close_device(ctx, dev);
    }
    fz_always(ctx)
    {
        // Throw away the device.
        fz_drop_device(ctx, dev);

        // The page is no longer needed, all drawing commands

```

(continues on next page)

(continued from previous page)

```

        // are now in the display list.
        fz_drop_page(ctx, page);
    }
    fz_catch(ctx)
        fz_rethrow(ctx);

    // Populate the data structure to be sent to the
    // rendering thread for this page.
    data = malloc(sizeof(*data));

    data->pagenumber = i + 1;
    data->ctx = ctx;
    data->list = list;
    data->bbox = bbox;
    data->pix = NULL;
    data->failed = 0;

    // Create the thread and pass it the data structure.
    if (pthread_create(&thread[i], NULL, renderer, data) != 0)
        fail("pthread_create()");
}

// Now each thread is rendering pages, so wait for each thread
// to complete its rendering.
fprintf(stderr, "joining %d threads...\n", threads);
for (i = 0; i < threads; i++)
{
    char filename[42];
    struct thread_data *data;

    if (pthread_join(thread[i], (void **) &data) != 0)
        fail("pthread_join");

    if (data->failed)
    {
        fprintf(stderr, "\tRendering for page %d failed\n", i + 1);
    }
    else
    {
        sprintf(filename, "out%04d.png", i);
        fprintf(stderr, "\tSaving %s...\n", filename);

        // Write the rendered image to a PNG file
        fz_save_pixmap_as_png(ctx, data->pix, filename);
    }

    // Free the thread's pixmap and display list.
    fz_drop_pixmap(ctx, data->pix);
    fz_drop_display_list(ctx, data->list);

    // Free the data structure passed back and forth

```

(continues on next page)

(continued from previous page)

```

        // between the main thread and rendering thread.
        free(data);
    }
}
fz_always(ctx)
{
    // Free the thread structure
    free(thread);

    // Drop the document
    fz_drop_document(ctx, doc);
}
fz_catch(ctx)
    fail(fz_caught_message(ctx));

// Finally the main thread's context is freed.
fz_drop_context(ctx);

fprintf(stderr, "finally!\n");
fflush(NULL);

return 0;
}

```

## 2.5 Cloning the context

As described above, every context contains an exception stack which is manipulated during the course of nested `fz_try` / `fz_catches`. For obvious reasons the same exception stack cannot be used from more than one thread at a time.

If, however, we simply created a new context (using `fz_new_context`) for every thread, we would end up with separate stores/glyph caches etc., which is not (generally) what is desired. *MuPDF* therefore provides a mechanism for “cloning” a context. This creates a new context that shares everything with the given context, except for the exception stack.

A commonly used general scheme is therefore to create a ‘base’ context at program start up, and to clone this repeatedly to get new contexts that can be used on new threads.

## 2.6 Coding Style

### 2.6.1 Names

Functions should be named according to one of the following schemes:

- **verb\_noun.**
- **verb\_noun\_with\_noun.**
- **noun\_attribute.**
- **set\_noun\_attribute.**
- **noun\_from\_noun** - convert from one type to another (avoid **noun\_to\_noun**).

Prefixes are mandatory for exported functions, macros, enums, globals and types:

- `fz` for common code.
- `pdf`, `xps`, etc., for interpreter specific code.

Prefixes are optional (but encouraged) for private functions and types.

Avoid using ‘get’ as this is a meaningless and redundant filler word.

These words are reserved for reference counting schemes:

- **new**, **create**, **find**, **load**, **open**, **keep** - return objects that you are responsible for freeing.
- **drop** - relinquish ownership of the object passed in.

When searching for an object or value, the name used depends on whether returning the value is passing ownership:

- **lookup** - return a value or borrowed pointer.
- **find** - return an object that the caller is responsible for freeing.

## 2.6.2 Types

Various different integer types are used throughout *MuPDF*.

In general:

- `int` is assumed to be 32bit at least.
- `short` is assumed to be exactly 16 bits.
- `char` is assumed to be exactly 8 bits.
- Array sizes, string lengths, and allocations are measured using `size_t`. `size_t` is 32bit in 32bit builds, and 64bit on all 64bit builds.
- Buffers of data use unsigned chars (or `uint8_t`).
- Offsets within files/streams are represented using `int64_t`.

In addition, we use floats (and avoid doubles when possible), assumed to be [IEEE compliant](#).

## 2.6.3 Reference counting

Reference counting uses special words in functions to make it easy to remember and follow the rules.

Words that take ownership: [new](#), [find](#), [load](#), `open`, `keep`.

Words that release ownership: `drop`.

If an object is returned by a function with one of the special words that take ownership, you are responsible for freeing it by calling `drop` or `free`, or [close](#) before you return. You may pass ownership of an owned object by return it only if you name the function using one of the special words.

Any objects returned by functions that do not have any of these special words, are borrowed and have a limited life time. Do not hold on to them past the duration of the current function, or stow them away inside structs. If you need to keep the object for longer than that, you have to either `keep` it or make your own copy.





## MUPDF ON THE COMMAND LINE

*MuPDF* has few command line options available:

- *mupdf-gl* A handy viewer UI.
- *muraster* Can be used to convert *PDF* pages to raster images.
- *mutool* An all purpose tool for dealing with *PDF* files. Used in conjunction with the *mutool JavaScript API*

### 3.1 *mupdf-gl*

The *OpenGL* based viewer can read *PDF*, *XPS*, *CBZ*, *EPUB*, & *FB2* documents as well as typical image formats including *SVG*. It compiles on any platform that has a *GLUT* library. The latest release builds on *Linux*, *Windows*, and *MacOS*.

#### 3.1.1 Command Line Options

`mupdf-gl [options] document [page]`

[options]

**-p password**

The password needed to open a password protected *PDF* file.

**-r resolution**

Set the initial zoom level, specified as DPI. The default value is 72.

**-W width**

Set the page width in points for *EPUB* layout.

**-H height**

Set the page height in points for *EPUB* layout.

**-S size**

Set the default font size in points for *EPUB* layout.

**-U stylesheet**

Specify a *CSS* file containing user styles to load for *EPUB* layout.

**-X**

Ignore publisher styles for *EPUB* layout.

**-J**

Disable *JavaScript* in *PDF* forms.

**-A level**

Set anti-aliasing level or pixel rendering rule.

- 0: off.
- 2: 4 levels.
- 4: 16 levels.
- 8: 256 levels.
- 9: using centre of pixel rule.
- 10: using any part of a pixel rule.

**-I**

Start in inverted color mode.

**-B hex-color**

Set black tint color (default 303030).

**-C hex-color**

Set white tint color (default FFFF0).

**-Y factor**

Set UI scaling factor (default calculated from screen DPI).

**[page]**

The initial page number to show.

**Example usage #1:**

This will open the viewer with a file browser which allows you to choose the file you need.

```
mupdf-gl
```

**Example usage #1:**

This will open a specific file, in inverted color mode, on page 10.

```
mupdf-gl -I mupdf_explored.pdf 10
```

## 3.1.2 Mouse Bindings

The middle mouse button (scroll wheel button) pans the document view.

The right mouse button selects a region and copies the marked text to the clipboard.

## 3.1.3 Key Bindings

Several commands can take a number argument entered before the key, to modify the command. For example, to zoom to 150 dpi, type “150z”.

<b>F1</b>	Display help.
<b>i</b>	Show document information.
<b>o</b>	Show document outline.
<b>L</b>	Highlight links.
<b>F</b>	Highlight form fields.
<b>a</b>	Show annotation editor.
<b>r</b>	Reload document.
<b>S</b>	Save document (only for <i>PDF</i> ).
<b>q</b>	Quit viewer.

<b>&lt;</b>	Decrease E-book font size.
<b>&gt;</b>	Increase E-book font size.
<b>I</b>	Toggle inverted color mode.
<b>C</b>	Toggle tinted color mode.
<b>E</b>	Toggle ICC color management.
<b>e</b>	Toggle spot color emulation.
<b>A</b>	Toggle anti-aliasing.

<b>f</b>	Toggle fullscreen.
<b>w</b>	Shrinkwrap window to fit page.
<b>W</b>	Fit page width to window.
<b>H</b>	Fit page height to window.
<b>Z</b>	Fit page size to window.
<b>[number] z</b>	Set zoom resolution in DPI.
<b>+</b>	Zoom in.
<b>-</b>	Zoom out.
<b>[</b>	Rotate counter-clockwise.
<b>]</b>	Rotate clockwise.
<b>or h, j, k, l</b>	Pan page in small increments.

<b>b</b>	Smart move one screenful backward.
<b>[space]</b>	Smart move one screenful forward.
<b>[comma] or [page up]</b>	Go one page backward.
<b>[period] or [page down]</b>	Go one page forward.
<b>[number] g</b>	Go to page number.
<b>G</b>	Go to last page.

<b>m</b>	Save current page to navigation history.
<b>t</b>	Go back in navigation history.
<b>T</b>	Go forward in navigation history.
<b>[number] m</b>	Save current page in numbered bookmark.
<b>[number] t</b>	Go to numbered bookmark.

<b>/</b>	Start searching forward.
<b>?</b>	Start searching backward.
<b>n</b>	Continue searching forward.
<b>N</b>	Continue searching backward.

## 3.2 *muraster*

This is a much simpler version of `mutool draw` command. As such it can be used to quickly rasterize an input file with a set of options.

### 3.2.1 Command Line Options

<code>muraster [options] file [pages]</code>
--

[options]

**-p password**

The password needed to open a password protected *PDF* file.

**-o filename**

The output file name.

**-F format**

The output format (default inferred from output file name), e.g. `pam`, `pbm`, `pgm`, `pkm`, `ppm`.

**-s information**

Show extra information: `-m`: show memory use. `-t`: show timings.

**-R rotation**

Set a rotation for the output (default `auto`): `-auto`. `-0`. `-90`. `-180`. `-270`. `-clockwise`.

**-r x,y**

Comma separated x and y resolution in DPI (default: 300,300).

**-w width**

Printable width (in inches) (default: 8.27).

**-h height**

Printable height (in inches) (default: 11.69).

**-f**

Fit file to page if too large.

**-B height**

Set the minimum band height (e.g. 32).

**-M memory**

Sets the maximum band memory (e.g. 655360).

**-W width**

Page width for EPUB layout.

**-H height**

Page height for EPUB layout.

**-S size**

Font size for EPUB layout.

**-U filename**

File name of user stylesheet for EPUB layout

**-X**

Disable document styles for EPUB layout

**-A level**

Set anti-aliasing level.

- 0: off.
- 2: 4 levels.
- 4: 16 levels.
- 8: 256 levels.

**-A graphics level / text level**

Independently set the anti-aliasing levels for graphics and text.

e.g. -A 0/4.

**[pages]**

A comma separated list of page numbers and ranges.

**Example usage #1:**

This will render a raster file from page one of the input file “mupdf\_explored.pdf”. The output file will be called “test.ppm”, have a clockwise rotation and specific graphics/text anti-aliasing applied.

```
muraster -o test.ppm -R clockwise -A 0/8 mupdf_explored.pdf 1
```

## 3.3 mutool

**Note:** It is advised to use [rlwrap](#) with `mutool` for command line history and cursor navigation (this can be also installed via *Homebrew* or *MacPorts*).

For rendering and converting documents there are three commands available:

### 3.3.1 mutool draw

The *draw* command will render a document to image files, convert to another vector format, or extract the text content.

- The supported input document formats are: pdf, xps, cbz, and epub.
- The supported output image formats are: pbm, pgm, ppm, pam, png, pwg, pcl and ps.
- The supported output vector formats are: svg, pdf, and debug trace (as xml).
- The supported output text formats are: plain text, html, and structured text (as xml or json).

```
mutool draw [options] file [pages]
```

**Note:** Command line parameters within square brackets [] are optional.

**[options]**

Options are as follows:

**-p password**

Use the specified password if the file is encrypted.

**-o output**

The output file name. The output format is inferred from the output filename. Embed %d in the name to indicate the page number (for example: “page%d.png”). Printf modifiers are supported, for example “%03d”. If no output is specified, the output will go to `stdout`.

**-F format**

Enforce a specific output format. Only necessary when outputting to `stdout` since normally the output filename is used to infer the output format.

**-R angle**

Rotate clockwise by given number of degrees.

**-r resolution**

Render the page at the specified resolution. The default resolution is 72 dpi.

**-w width**

Render the page at the specified width (or, if the `-r` flag is used, render with a maximum width).

**-h height**

Render the page at the specified height (or, if the `-r` flag is used, render with a maximum height).

**-f**

Fit exactly; ignore the aspect ratio when matching specified width/heights.

**-B bandheight**

Render in banded mode with each band no taller than the given height. This uses less memory during rendering. Only compatible with `pam`, `pgm`, `ppm`, `pnm` and `png` output formats. Banded rendering and md5 checksumming may not be used at the same time.

**-W width**

Page width in points for *EPUB* layout.

**-H height**

Page height in points for *EPUB* layout.

**-S size**

Font size in points for *EPUB* layout.

**-U filename**

User CSS stylesheet for *EPUB* layout.

**-X**

Disable document styles for *EPUB* layout.

**-c colorspace**

Render in the specified colorspace. Supported colorspace are: `mono`, `gray`, `grayalpha`, `rgb`, `rgba`, `cmyk`, `cmykalpha`. Some abbreviations are allowed: `m`, `g`, `ga`, `rgba`, `cmyka`. The default is chosen based on the output format.

**-G gamma**

Apply gamma correction. Some typical values are 0.7 or 1.4 to thin or darken text rendering.

**-I**

Invert colors.

**-s [mft5]**

Show various bits of information: `m` for glyph cache and total memory usage, `f` for page features such as whether the page is grayscale or color, `t` for per page rendering times as well statistics, and `5` for md5 checksums of rendered images that can be used to check if rendering has changed.

**-A bits**

Specify how many bits of anti-aliasing to use. The default is 8. 0 means no anti-aliasing, 9 means no anti-aliasing, centre-of-pixel rule, 10 means no anti-aliasing, any-part-of-a-pixel rule.

**-D**

Disable use of display lists. May cause slowdowns, but should reduce the amount of memory used.

**-i**

Ignore errors.

**-L**

Low memory mode (avoid caching objects by clearing cache after each page).

**-P**

Run interpretation and rendering at the same time.

---

**file**

Input file name. The input can be any of the *document formats supported by MuPDF*.

---

**[pages]**

Comma separated list of page ranges. The first page is “1”, and the last page is “N”. The default is “1-N”.

This is the more customizable tool, but also has a more difficult set of command line options. It is primarily used for rendering a document to image files.

### 3.3.2 *mutool convert*

The `convert` command converts an input file into another format.

```
mutool convert [options] file [pages]
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

**[options]**

Options are as follows:

**-p password**

Use the specified password if the file is encrypted.

**-o output**

The output file name. The output format is inferred from the output filename. Embed %d in the name to indicate the page number (for example: “page%d.png”). Printf modifiers are supported, for example “%03d”. If no output is specified, the output will go to `stdout`.

**-F output format (default inferred from output file name)**

- raster: cbz, png, pnm, pgm, ppm, pam, pbm, pkm.
- print-raster: pcl, pclm, ps, pwg.
- vector: pdf, svg.

- text: html, xhtml, text, stext.

**-A bits**

Specify how many bits of anti-aliasing to use. The default is 8.

**-W width**

Page width in points for *EPUB* layout.

**-H height**

Page height in points for *EPUB* layout.

**-S size**

Font size in points for *EPUB* layout.

**-U filename**

User CSS stylesheet for *EPUB* layout.

**-X**

Disable document styles for *EPUB* layout.

-O comma separated list of options for output format.

**Raster output options:**

- rotate=N Rotate rendered pages N degrees counterclockwise.
- resolution=N Set both X and Y resolution in pixels per inch.
- x-resolution=N X resolution of rendered pages in pixels per inch.
- y-resolution=N Y resolution of rendered pages in pixels per inch.
- width=N Render pages to fit N pixels wide (ignore resolution option).
- height=N Render pages to fit N pixels tall (ignore resolution option).
- colorspace=(gray|rgb|cmyk) Render using specified colorspace.
- alpha Render pages with alpha channel and transparent background.
- graphics=(aaN|cop|app) Set the rasterizer to use for graphics.
  - aaN Antialias with N bits (0 to 8).
  - cop Center of pixel.
  - app Any part of pixel.
- text=(aaN|cop|app) Set the rasterizer to use for text.
  - aaN Antialias with N bits (0 to 8).
  - cop Center of pixel.
  - app Any part of pixel.

**PCL output options:**

- colorspace=mono Render 1-bit black and white page.
- colorspace=rgb Render full color page.
- preset=generic|ljet4|dj500|fs600|lj|lj2|lj3|lj3d|lj4|lj4pl|lj4d|lp2563b|oce9050.
- spacing=0 No vertical spacing capability.
- spacing=1 PCL 3 spacing (<ESC>\*p+<n>Y).
- spacing=2 PCL 4 spacing (<ESC>\*b+<n>Y).



- `spacing=3` PCL 5 spacing (`<ESC>*b<n>Y` and clear seed row).
- `mode2` Enable mode 2 graphics compression.
- `mode3` Enable mode 3 graphics compression.
- `eog_reset` End of graphics (`<ESC>*rB`) resets all parameters.
- `has_duplex` Duplex supported (`<ESC>&l<duplex>S`).
- `has_papersize` Papersize setting supported (`<ESC>&l<sizecode>A`).
- `has_copies` Number of copies supported (`<ESC>&l<copies>X`).
- `is_ljet4pjl` Disable/Enable HP 4PJM model-specific output.
- `is_ace9050` Disable/Enable Ace 9050 model-specific output.

**PCLm output options:**

- `compression=none` No compression (default).
- `compression=flate` Flate compression.
- `strip-height=N` Strip height (default 16).

**PWG output options:**

- `media_class=` Set the `media_class` field.
- `media_color=` Set the `media_color` field.
- `media_type=` Set the `media_type` field.
- `output_type=` Set the `output_type` field.
- `rendering_intent=` Set the `rendering_intent` field.
- `page_size_name=` Set the `page_size_name` field.
- `advance_distance=` Set the `advance_distance` field.
- `advance_media=` Set the `advance_media` field.
- `collate=` Set the `collate` field.
- `cut_media=` Set the `cut_media` field.
- `duplex=` Set the `duplex` field.
- `insert_sheet=` Set the `insert_sheet` field.
- `jog=` Set the `jog` field.
- `leading_edge=` Set the `leading_edge` field.
- `manual_feed=` Set the `manual_feed` field.
- `media_position=` Set the `media_position` field.
- `media_weight=` Set the `media_weight` field.
- `mirror_print=` Set the `mirror_print` field.
- `negative_print=` Set the `negative_print` field.
- `num_copies=` Set the `num_copies` field.
- `orientation=` Set the `orientation` field.
- `output_face_up=` Set the `output_face_up` field.

- `page_size_x`= Set the `page_size_x` field.
- `page_size_y`= Set the `page_size_y` field.
- `separations`= Set the `separations` field.
- `tray_switch`= Set the `tray_switch` field.
- `tumble`= Set the `tumble` field.
- `media_type_num`= Set the `media_type_num` field.
- `compression`= Set the `compression` field.
- `row_count`= Set the `row_count` field.
- `row_feed`= Set the `row_feed` field.
- `row_step`= Set the `row_step` field.

**Text output options:**

- `inhibit-spaces` Don't add spaces between gaps in the text.
- `preserve-images` Keep images in output.
- `preserve-ligatures` Do not expand ligatures into constituent characters.
- `preserve-whitespace` Do not convert all whitespace into space characters.
- `preserve-spans` Do not merge spans on the same line.
- `dehyphenate` Attempt to join up hyphenated words.
- `mediabox-clip=no` Include characters outside mediabox.

**PDF output options:**

- `decompress` Decompress all streams (except `compress-fonts/images`).
- `compress` Compress all streams.
- `compress-fonts` Compress embedded fonts.
- `compress-images` Compress images.
- `ascii` ASCII hex encode binary streams.
- `pretty` Pretty-print objects with indentation.
- `linearize` Optimize for web browsers.
- `clean` Pretty-print graphics commands in content streams.
- `sanitize` Sanitize graphics commands in content streams.
- `incremental` Write changes as incremental update.
- `continue-on-error` Continue saving the document even if there is an error.
- `garbage` Garbage collect unused objects.
  - or `garbage=compact` ... and compact cross reference table.
  - or `garbage=deduplicate` ... and remove duplicate objects.
- `decrypt` Write unencrypted document.
- `encrypt=rc4-40|rc4-128|aes-128|aes-256` Write encrypted document.
- `permissions=NUMBER` Document permissions to grant when encrypting.

- `user-password=PASSWORD` Password required to read document.
- `owner-password=PASSWORD` Password required to edit document.
- `regenerate-id` Regenerate document id (default yes).

**SVG output options:**

- `text=text` Emit text as `<text>` elements (inaccurate fonts).
  - `text=path` Emit text as `<path>` elements (accurate fonts).
  - `no-reuse-images` Do not reuse images using `<symbol>` definitions.
- 

**file**

Input file name. The input can be any of the *document formats supported by MuPDF*.

---

**[pages]**

Comma separated list of page ranges. The first page is “1”, and the last page is “N”. The default is “1-N”.

This tool is used for converting documents into other formats, and is easier to use.

### 3.3.3 *mutool trace*

The trace command prints a trace of device calls needed to render a page.

```
mutool trace [options] file [pages]
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

**[options]**

Options are as follows:

- p password**  
Use the specified password if the file is encrypted.
- W width**  
Page width in points for *EPUB* layout.
- H height**  
Page height in points for *EPUB* layout.
- S size**  
Font size in points for *EPUB* layout.
- U filename**  
User CSS stylesheet for *EPUB* layout.
- X**  
Disable document styles for *EPUB* layout.
- d**  
Use display list.

**file**

Input file name. The input can be any of the *document formats supported by MuPDF*.

---

**[pages]**

Comma separated list of page ranges. The first page is “1”, and the last page is “N”. The default is “1-N”.

---

The trace takes the form of an XML document, with the root element being the document, its children each page, and one page child element for each device call on that page.

**An example trace:**

```
<document filename="hello.pdf">
<page number="1" mediabox="0 0 595 842">
<fill_path winding="nonzero" colorspace="DeviceRGB" color="1 0 0" matrix="1 0 0
↪ 0 -1 0 842">
<moveto x="50" y="50"/>
<lineto x="100" y="200"/>
<lineto x="200" y="50"/>
</fill_path>
<fill_text colorspace="DeviceRGB" color="0" matrix="1 0 0 -1 0 842">
<span font="Times-Roman" wmode="0" trm="100 0 0 100">
<g unicode="H" glyph="H" x="50" y="500" />
<g unicode="e" glyph="e" x="122.2" y="500" />
<g unicode="l" glyph="l" x="166.6" y="500" />
<g unicode="l" glyph="l" x="194.4" y="500" />
<g unicode="o" glyph="o" x="222.2" y="500" />
<g unicode="!" glyph="exclam" x="272.2" y="500" />
</span>
</fill_text>
</page>
</document>
```

This is a debugging tool used for printing a trace of the graphics device calls on a page.

---

There are also several tools specifically for working with *PDF* files:

### 3.3.4 *mutool show*

The show command will print the specified objects and streams to stdout. Streams are decoded and non-printable characters are represented with a period by default.

```
mutool show [options] input.pdf ( trailer | xref | pages | grep | outline | js_
↪ | form | <path> ) *
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

**[options]**

Options are as follows:

**-p password**

Use the specified password if the file is encrypted.

**-o output**

The output file name instead of using stdout. Should be a plain text file format.

**-e**

Leave stream contents in their original form.

**-b**

Print only stream contents, as raw binary data.

**-g**

Print only object, one line per object, suitable for grep.

**input.pdf**

Input file name. Must be a *PDF* file.

( **trailer** | **xref** | **pages** | **grep** | **outline** | **js** | **form** | **<path>** ) \*

Specify what to show by using one of the following keywords, or specify a path to an object:

**trailer**

Print the trailer dictionary.

**xref**

Print the cross reference table.

**pages**

List the object numbers for every page.

**grep**

Print all the objects in the file in a compact one-line format suitable for piping to grep.

**outline**

Print the outline (also known as “table of contents” or “bookmarks”).

**js**

Print document level *JavaScript*.

**form**

Print form objects.

**<path>**

A path starts with either an object number, a property in the trailer dictionary, or the keyword “trailer” or “pages”. Separate elements with a period ‘.’ or slash ‘/’. Select a page object by using pages/N where N is the page number. The first page is number 1.

\*

You can use \* as an element to iterate over all array indices or dictionary properties in an object. Thus you can have multiple keywords with for your `mutool show` query.

**Examples:**

Find the number of pages in a document:

```
mutool show $FILE trailer/Root/Pages/Count
```

Print the raw content stream of the first page:

```
mutool show -b $FILE pages/1/Contents
```

Print the raw content stream of the first page & second page & the PDF outline (demonstrates use of the \* element):

```
mutool show -b $FILE pages/1/Contents pages/2/Contents outline
```

Show all *JPEG* compressed stream objects:

```
mutool show $FILE grep | grep '/Filter/DCTDecode'
```

A tool for displaying the internal objects in a *PDF* file.

### 3.3.5 mutool extract

The **extract** command can be used to extract images and font files from a *PDF* file. The image and font files will be saved out to the same folder which the file originates from.

```
mutool extract [options] input.pdf [object numbers]
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

#### [options]

Options are as follows:

- p password**  
Use the specified password if the file is encrypted.
- r**  
Convert images to RGB when extracting them.
- a**  
Embed SMasks as alpha channel.
- N**  
Do not use ICC color conversions.

---

#### input.pdf

Input file name. Must be a *PDF* file.

---

#### [object numbers]

An array of object ids to extract from. If no object numbers are given on the command line, all images and fonts will be extracted from the file.

Extract images and embedded font resources.

### 3.3.6 *mutool clean*

The `clean` command pretty prints and rewrites the syntax of a *PDF* file. It can be used to repair broken files, expand compressed streams, filter out a range of pages, etc.

```
mutool clean [options] input.pdf [output.pdf] [pages]
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

#### [options]

Options are as follows:

- p password**  
Use the specified password if the file is encrypted.
- g**  
Garbage collect unused objects.
- gg**  
In addition to `-g` compact xref table.
- ggg**  
In addition to `-gg` merge duplicate objects.
- gggg**  
In addition to `-ggg` check streams for duplication.
- l**  
Linearize PDF.
- D**  
Save file without encryption.
- E encryption**  
Save file with new encryption (rc4-40, rc4-128, aes-128, or aes-256).
- O owner\_password**  
Owner password (only if encrypting).
- U user\_password**  
User password (only if encrypting).
- P permission**  
Permission flags (only if encrypting).
- a**  
ASCII hex encode binary streams.
- d**  
Decompress streams.
- z**  
Deflate uncompressed streams.
- f**  
Compress font streams.
- i**  
Compress image streams.

- c**  
Clean content streams.
  - s**  
Sanitize content streams.
  - A**  
Create appearance streams for annotations.
  - AA**  
Recreate appearance streams for annotations.
  - m**  
Preserve metadata.
- 

**input.pdf**

Input file name. Must be a *PDF* file.

---

**[output.pdf]**

The output file. Must be a *PDF* file.

---

**Note:** If no output file is specified, it will write the cleaned *PDF* to “out.pdf” in the current directory.

---

**[pages]**

Comma separated list of page numbers and ranges. If no pages are supplied then all document pages will be considered for the output file.

Rewrite *PDF* files. Used to fix broken files, or to make a *PDF* file human editable.

### 3.3.7 mutool merge

The merge command is used to pick out pages from two or more files and merge them into a new output file.

```
mutool merge [-o output.pdf] [-O options] input.pdf [pages] [input2.pdf] ↵  
↪ [pages2] ...
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

**[-o output.pdf]**

The output filename. Defaults to “out.pdf” if not supplied.

---

**[-O options]**

Comma separated list of output options.

- **decompress** Decompress all streams (except compress-fonts/images).



- `compress` Compress all streams.
- `compress-fonts` Compress embedded fonts.
- `compress-images` Compress images.
- `ascii` ASCII hex encode binary streams.
- `pretty` Pretty-print objects with indentation.
- `linearize` Optimize for web browsers.
- `clean` Pretty-print graphics commands in content streams.
- `sanitize` Sanitize graphics commands in content streams.
- `incremental` Write changes as incremental update.
- `continue-on-error` Continue saving the document even if there is an error.
- `garbage` Garbage collect unused objects.
  - or `garbage=compact ...` and compact cross reference table.
  - or `garbage=deduplicate ...` and remove duplicate objects.
- `decrypt` Write unencrypted document.
- `encrypt=rc4-40|rc4-128|aes-128|aes-256` Write encrypted document.
- `permissions=NUMBER` Document permissions to grant when encrypting.
- `user-password=PASSWORD` Password required to read document.
- `owner-password=PASSWORD` Password required to edit document.
- `regenerate-id` Regenerate document id (default yes).

---

**input.pdf**

The first document.

---

**[pages]**

Comma separated list of page ranges for the first document (`input.pdf`). The first page is “1”, and the last page is “N”. The default is “1-N”.

---

**[input2.pdf]**

The second document.

---

**[pages2]**

Comma separated list of page ranges for the second document (`input2.pdf`). The first page is “1”, and the last page is “N”. The default is “1-N”.

---

...

Indicates that we add as many additional `[input]` & `[pages]` pairs as required to merge multiple documents.

Merge pages from multiple input files into a new *PDF*.

### 3.3.8 *mutool poster*

The `poster` command reads the input PDF file and for each page chops it up into `x` by `y` pieces. Each piece becomes its own page in the output *PDF* file. This makes it possible for each page to be printed upscaled and can then be merged into a large poster.

```
mutool poster [options] input.pdf [output.pdf]
```

---

**Note:** Command line parameters within square brackets `[]` are optional.

---

#### [options]

Options are as follows:

**-p password**

Use the specified password if the file is encrypted.

**-x x decimation factor**

Pieces to horizontally divide each page into.

**-y y decimation factor**

Pieces to vertically divide each page into.

---

#### **input.pdf**

The input *PDF* document.

---

#### [output.pdf]

The output filename. Defaults to “out.pdf” if not supplied.

Divide pages of a *PDF* into pieces that can be printed and merged into a large poster.

### 3.3.9 *mutool create*

The `create` command creates a new *PDF* file with the contents created from one or more input files containing graphics commands.

```
mutool create [-o output.pdf] [-O options] page.txt [page2.txt ...]
```

---

**Note:** Command line parameters within square brackets `[]` are optional.

---

#### **[-o output.pdf]**

The output filename. Defaults to “out.pdf” if not supplied.

---

#### **[-O options]**

The `-O` argument is a comma separated list of options for writing the *PDF* file:

- **decompress** Decompress all streams (except compress-fonts/images).
  - **compress** Compress all streams.
-

- `compress-fonts` Compress embedded fonts.
- `compress-images` Compress images.
- `ascii` ASCII hex encode binary streams.
- `pretty` Pretty-print objects with indentation.
- `linearize` Optimize for web browsers.
- `clean` Pretty-print graphics commands in content streams.
- `sanitize` Sanitize graphics commands in content streams.
- `incremental` Write changes as incremental update.
- `continue-on-error` Continue saving the document even if there is an error.
- `garbage` Garbage collect unused objects.
  - or `garbage=compact ...` and compact cross reference table.
  - or `garbage=deduplicate ...` and remove duplicate objects.
- `decrypt` Write unencrypted document.
- `encrypt=rc4-40|rc4-128|aes-128|aes-256` Write encrypted document.
- `permissions=NUMBER` Document permissions to grant when encrypting.
- `user-password=PASSWORD` Password required to read document.
- `owner-password=PASSWORD` Password required to edit document.
- `regenerate-id` Regenerate document id (default yes).

---

**page.txt**

*Page content stream* with annotations for creating resources.

---

**[page2.txt ...]**

Define multiple *page content streams* as required.

---

## Page content streams

A page is created for each input file, with the contents of the file copied into the content stream. Special comments in the input files are parsed to define the page dimensions and font and image resources.

### Example #1

Define an image:

```
%%MediaBox 0 0 500 800
%%Rotate 90
%%Image Im0 path/to/image.png
```

### Example #2

Font resources can be created by either giving the name of a standard *PDF* font, or by giving the path to a font file. If a third argument is present and either “Greek” or “Cyrillic” the font will be encoded using ISO 8859-7 or KOI8-U, respectively:

```
%%Font Tm Times-Roman
%%Font TmG Times-Roman Greek
%%Font TmC Times-Roman Cyrillic
%%Font Fn0 path/to/font/file.ttf
%%Font Fn1 path/to/font/file.ttf Cyrillic
```

### Example #3

CJK fonts can be created by passing a language tag for one of the 4 CID orderings: zh-Hant, zh-Hans, ja, or ko (Traditional Chinese, Simplified Chinese, Japanese, Korean). The CJK font will use the UTF-16 encoding. A font file will not be embedded, so a *PDF* viewer will use a substitute font:

```
%%CJKFont Batang ko
%%CJKFont Mincho ja
%%CJKFont Ming zh-Hant
%%CJKFont Song zh-Hans
```

### Example #4

An example input file, which adds an image, a triangle and some text:

```
%%MediaBox 0 0 595 842
%%Font TmRm Times-Roman
%%Font Helv-C Helvetica Cyrillic
%%Font Helv-G Helvetica Greek
%%CJKFont Song zh-Hant
%%CJKFont Mincho ja
%%CJKFont Batang ko
%%Image I0 logo/mupdf-simplified-logo.png

% Draw an image.
q
480 0 0 480 50 250 cm
/I0 Do
Q

% Draw a triangle.
q
1 0 0 rg
50 50 m
100 200 l
200 50 l
f
Q

% Show some text.
q
0 0 1 rg
BT /TmRm 24 Tf 50 760 Td (Hello, world!) Tj ET
BT /Helv-C 24 Tf 50 730 Td <fac4d2c1d7d3d4d7d5cad4c521> Tj ET
BT /Helv-G 24 Tf 50 700 Td <eae1ebe7ecddf1e1> Tj ET
BT /Song 24 Tf 50 670 Td <4F60 597D> Tj ET
BT /Mincho 24 Tf 50 640 Td <3053 3093 306b 3061 306f> Tj ET
BT /Batang 24 Tf 50 610 Td <c548 b155 d558 c138 c694> Tj ET
```

(continues on next page)

(continued from previous page)

Q

Create a new *PDF* file from a text file with graphics commands.

### 3.3.10 *mutool sign*

The *sign* command reads an input *PDF* file and by default prints information about each signature field object. The command applies to the signature field objects given, or every signature field if none are specified. Given suitable option signature fields can be verified, cleared or signed using a given certificate and certificate password.

```
mutool sign [options] input.pdf [signature object numbers]
```

**Note:** Command line parameters within square brackets [] are optional.

#### [options]

Options are as follows:

**-p password**

Use the specified password if the file is encrypted.

**-v**

Verify each signature field and check whether the document has changed since signing.

**-c**

Revert each signed signature field back to its unsigned state.

**-s certificate file**

Sign each signature field with the certificate in the given file.

**-P certificate password**

The password used together with the certificate to sign a signature field.

**-o filename**

Output *PDF* file name.

#### input.pdf

The input *PDF* document.

#### [signature object numbers]

Can be used to specify a particular set of signature field objects to apply the *sign* command to.

## Signing certificates

Signing digital signatures in *MuPDF* requires that you have a PFX certificate. You can create a self-signed certificate using *OpenSSL* by following these steps:

- 1) Generate a self-signed certificate and private key:

```
$ openssl req -x509 -days 365 -newkey rsa:2048 -keyout cert.pem -out cert.pem
```

- 2) Convert to PFX format:

```
$ openssl pkcs12 -export -in cert.pem -out cert.pfx
```

List, verify, and sign digital signatures in *PDF* files.

### 3.3.11 *mutool info*

The `info` command shows info about the objects on pages in an input file. For each page the page object number is shown, and then detailed information about the desired objects.

```
mutool info [options] file [pages]
```

---

**Note:** Command line parameters within square brackets [] are optional.

---

#### [options]

Options are as follows:

- p password**  
Use the specified password if the file is encrypted.
- F**  
List fonts.
- I**  
List images.
- M**  
List dimensions.
- P**  
List patterns.
- S**  
List shadings.
- X**  
List form and postscript xobjects.

---

#### file

The input file.

---

#### [pages]

Comma separated list of page numbers and ranges. If no pages are supplied then all document pages will be considered for the output file.

Prints details about objects on each page in a *PDF* file.

### 3.3.12 *mutool pages*

The `pages` command prints out `MediaBox`, `CropBox`, etc. as well as `Rotation` and `UserUnit` for each page given (or all if not specified).

```
mutool pages [options] input.pdf [pages]
```

---

**Note:** Command line parameters within square brackets `[]` are optional.

---

#### [options]

Options are as follows:

**-p password**

Use the specified password if the file is encrypted.

---

#### input.pdf

Input file name. Must be a *PDF* file.

---

#### [pages]

Comma separated list of page numbers and ranges. If no pages are supplied then all document pages will be considered for the output file.

Prints details about media box, artbox, etc. for each page in a *PDF* file.

### 3.3.13 *mutool run*

The `run` command executes a *JavaScript* program, which has access to most of the features of the *MuPDF* library. The command supports *ECMAScript* 5 syntax in strict mode. All of the *MuPDF* constructors and functions live in the global object, and the command line arguments are accessible from the global `scriptArgs` object. The name of the script is in the global `scriptPath` variable.

See the *mutool JavaScript API* for more.

```
mutool run script.js [ arguments ... ]
```

---

**Note:** Command line parameters within square brackets `[]` are optional.

---

#### script.js

The *JavaScript* file which you would like to run.

---

**Note:** See the *mutool JavaScript API* for more.

---

[ arguments ... ]

If invoked without any arguments, it will drop you into an interactive *REPL* (read-eval-print-loop). To exit this loop type *quit()* (same as pressing *ctrl-C* two times).

## JavaScript Shell

Several global functions that are common for command line shells are available:

**gc(report)**

Run the garbage collector to free up memory. Optionally report statistics on the garbage collection.

**load(fileName)**

Load and execute script in “fileName”.

**print(...)**

Print arguments to stdout, separated by spaces and followed by a newline.

*quit()*

Exit the shell.

**read(fileName)**

Read the contents of a file and return them as a *UTF-8* decoded string.

*readline()*

Read one line of input from stdin and return it as a string.

**require(module)**

Load a *JavaScript* module.

**write(...)**

Print arguments to stdout, separated by spaces.

## Example scripts

### Create and edit *PDF* documents

#### Create *PDF* document from scratch using only low level functions

Listing 1: docs/examples/pdf-create-lowlevel.js

```
// Create a PDF from scratch.

// This example creates a new PDF file from scratch, using only the low level
↪ APIs.
// This assumes a basic working knowledge of the PDF file format.

// Create a new empty document with no pages.
var pdf = new PDFDocument()

// Create and add a font resource.
var font = pdf.addObject({
  Type: "Font",
  Subtype: "Type1",
  Encoding: "WinAnsiEncoding",
  BaseFont: "Times-Roman",
})
```

(continues on next page)



(continued from previous page)

```

// Create and add an image resource:
var image = pdf.addRawStream(
    // The raw stream contents, hex encoded to match the Filter entry:
    "004488CCEE7733>",
    // The image object dictionary:
    {
        Type: "XObject",
        Subtype: "Image",
        Width: 4,
        Height: 2,
        BitsPerComponent: 8,
        ColorSpace: "DeviceGray",
        Filter: "ASCIIHexDecode",
    }
);

// Create resource dictionary.
var resources = pdf.addObject({
    Font: { Tm: font },
    XObject: { Im0: image },
});

// Create content stream.
var buffer = new Buffer()
buffer.writeLine("10 10 280 330 re s")
buffer.writeLine("q 200 0 0 200 50 100 cm /Im0 Do Q")
buffer.writeLine("BT /Tm 16 Tf 50 50 TD (Hello, world!) Tj ET")
var contents = pdf.addStream(buffer)

// Create page object.
var page = pdf.addObject({
    Type: "Page",
    MediaBox: [0,0,300,350],
    Contents: contents,
    Resources: resources,
});

// Insert page object into page tree.
var pagetree = pdf.getTrailer().Root.Pages
pagetree.Count = 1
pagetree.Kids = [ page ]
page.Parent = pagetree

// Save the document.
pdf.save("out.pdf")

```

Create PDF document from scratch, using helper functions

Listing 2: docs/examples/pdf-create.js

```
// Create a PDF from scratch using helper functions.

// This example creates a new PDF file from scratch, using helper
// functions to create resources and page objects.
// This assumes a basic working knowledge of the PDF file format.

// Create a new empty document with no pages.
var pdf = new PDFDocument()

// Load built-in font and create WinAnsi encoded simple font resource.
var font = pdf.addSimpleFont(new Font("Times-Roman"))

// Load PNG file and create image resource.
var image = pdf.addImage(new Image("example.png"))

// Create resource dictionary.
var resources = pdf.addObject({
    Font: { Tm: font },
   XObject: { Im0: image },
})

// Create content stream data.
var contents =
    "10 10 280 330 re s\n" +
    "q 200 0 0 200 50 100 cm /Im0 Do Q\n" +
    "BT /Tm 16 Tf 50 50 TD (Hello, world!) Tj ET\n"

// Create a new page object.
var page = pdf.addPage([0,0,300,350], 0, resources, contents)

// Insert page object at the end of the document.
pdf.insertPage(-1, page)

// Save the document to file.
pdf.save("out.pdf", "pretty,ascii,compress-images,compress-fonts")
```

---

### Merge pages from multiple PDF documents into one PDF file

Listing 3: docs/examples/pdf-merge.js

```
// A re-implementation of "mutool merge" in JavaScript.

function copyPage(dstDoc, srcDoc, pageNumber, dstFromSrc) {
    var srcPage, dstPage
    srcPage = srcDoc.findPage(pageNumber)
    dstPage = dstDoc.newDictionary()
    dstPage.Type = dstDoc.newName("Page")
    if (srcPage.MediaBox) dstPage.MediaBox = dstFromSrc.
    ↪graftObject(srcPage.MediaBox)
    if (srcPage.Rotate) dstPage.Rotate = dstFromSrc.graftObject(srcPage.
```

(continues on next page)

(continued from previous page)

```

↪Rotate)
    if (srcPage.Resources) dstPage.Resources = dstFromSrc.
↪graftObject(srcPage.Resources)
    if (srcPage.Contents) dstPage.Contents = dstFromSrc.
↪graftObject(srcPage.Contents)
    dstDoc.insertPage(-1, dstDoc.addObject(dstPage))
}

function copyAllPages(dstDoc, srcDoc) {
    var dstFromSrc = dstDoc.newGraftMap()
    var k, n = srcDoc.countPages()
    for (k = 0; k < n; ++k)
        copyPage(dstDoc, srcDoc, k, dstFromSrc)
}

function pdfmerge() {
    var srcDoc, dstDoc, i

    dstDoc = new PDFDocument()
    for (i = 1; i < scriptArgs.length; ++i) {
        srcDoc = new PDFDocument(scriptArgs[i])
        copyAllPages(dstDoc, srcDoc)
    }
    dstDoc.save(scriptArgs[0], "compress")
}

if (scriptArgs.length < 2)
    print("usage: mutool run pdf-merge.js output.pdf input1.pdf input2.pdf ↪
↪...")
else
    pdfmerge()

```

## Graphics and the device interface

### Draw all pages in a document to PNG files

Listing 4: docs/examples/draw-document.js

```

// Draw all pages in a document and save them as PNG files.

var doc = new Document(scriptArgs[0]);
var n = doc.countPages();
for (var i = 0; i < n; ++i) {
    var page = doc.loadPage(i);
    var pixmap = page.toPixmap(Identity, DeviceRGB);
    pixmap.saveAsPNG("out" + (i+1) + ".png");
}

```

### Use device API to draw graphics and save as a PNG file

Listing 5: docs/examples/draw-device.js

```

// Use device interface to draw some graphics and save as a PNG.

var font = new Font("Times-Roman");
var image = new Image("example.png");
var path, text;

var pixmap = new Pixmap(DeviceRGB, [0,0,500,600], false);
pixmap.clear(255);
var device = new DrawDevice(Identity, pixmap);
var transform = [2,0,0,2,0,0]
{
    text = new Text();
    {
        text.showString(font, [16,0,0,-16,100,30], "Hello, world!");
        text.showString(font, [0,16,16,0,15,100], "Hello, world!");
    }
    device.fillText(text, transform, DeviceGray, [0], 1);

    path = new Path();
    {
        path.moveTo(10, 10);
        path.lineTo(90, 10);
        path.lineTo(90, 90);
        path.lineTo(10, 90);
        path.closePath();
    }
    device.fillPath(path, false, transform, DeviceRGB, [1,0,0], 1);
    device.strokePath(path, {dashes:[5,10], lineWidth:3, lineCap:'Round'},
→transform, DeviceRGB, [0,0,0], 1);

    path = new Path();
    {
        path.moveTo(100,100);
        path.curveTo(150,100, 200,150, 200,200);
        path.curveTo(200,300, 0,300, 100,100);
        path.closePath();
    }
    device.clipPath(path, true, transform);
    {
        device.fillImage(image, Concat(transform, [300,0,0,300,0,0]),
→1);
    }
    device.popClip();
}
device.close();

pixmap.saveAsPNG("out.png");

```

### Implement a device in JavaScript

Listing 6: docs/examples/trace-device.js

```

var Q = JSON.stringify

var pathPrinter = {
  moveTo: function (x,y) { print("moveTo", x, y) },
  lineTo: function (x,y) { print("lineTo", x, y) },
  curveTo: function (x1,y1,x2,y2,x3,y3) { print("curveTo", x1, y1, x2,
↪y2, x3, y3) },
  closePath: function () { print("closePath") },
}

var textPrinter = {
  beginSpan: function (f,m,wmode, bidi, dir, lang) {
    print("beginSpan",f,m,wmode,bidi,dir,repr(lang));
  },
  showGlyph: function (f,m,g,u,v,b) { print("glyph",f,m,g,u,v,b) },
  endSpan: function () { print("endSpan"); }
}

var traceDevice = {
  fillPath: function (path, evenOdd, ctm, colorSpace, color, alpha) {
    print("fillPath", evenOdd, ctm, colorSpace, color, alpha)
    path.walk(pathPrinter)
  },
  clipPath: function (path, evenOdd, ctm) {
    print("clipPath", evenOdd, ctm)
    path.walk(pathPrinter)
  },
  strokePath: function (path, stroke, ctm, colorSpace, color, alpha) {
    print("strokePath", Q(stroke), ctm, colorSpace, color, alpha)
    path.walk(pathPrinter)
  },
  clipStrokePath: function (path, stroke, ctm) {
    print("clipStrokePath", Q(stroke), ctm)
    path.walk(pathPrinter)
  },
  fillText: function (text, ctm, colorSpace, color, alpha) {
    print("fillText", ctm, colorSpace, color, alpha)
    text.walk(textPrinter)
  },
  clipText: function (text, ctm) {
    print("clipText", ctm)
    text.walk(textPrinter)
  },
  strokeText: function (text, stroke, ctm, colorSpace, color, alpha) {
    print("strokeText", Q(stroke), ctm, colorSpace, color, alpha)
    text.walk(textPrinter)
  },
  clipStrokeText: function (text, stroke, ctm) {
    print("clipStrokeText", Q(stroke), ctm)

```

(continues on next page)

(continued from previous page)

```

        text.walk(textPrinter)
    },
    ignoreText: function (text, ctm) {
        print("ignoreText", ctm)
        text.walk(textPrinter)
    },

    fillShade: function (shade, ctm, alpha) {
        print("fillShade", shade, ctm, alpha)
    },
    fillImage: function (image, ctm, alpha) {
        print("fillImage", image, ctm, alpha)
    },
    fillImageMask: function (image, ctm, colorSpace, color, alpha) {
        print("fillImageMask", image, ctm, colorSpace, color, alpha)
    },
    clipImageMask: function (image, ctm) {
        print("clipImageMask", image, ctm)
    },

    beginMask: function (area, luminosity, colorspace, color) {
        print("beginMask", area, luminosity, colorspace, color)
    },
    endMask: function () {
        print("endMask")
    },

    popClip: function () {
        print("popClip")
    },

    beginGroup: function (area, isolated, knockout, blendmode, alpha) {
        print("beginGroup", area, isolated, knockout, blendmode, alpha)
    },
    endGroup: function () {
        print("endGroup")
    },
    beginTile: function (area, view, xstep, ystep, ctm, id) {
        print("beginTile", area, view, xstep, ystep, ctm, id)
        return 0
    },
    endTile: function () {
        print("endTile")
    },

    close: function () {
        print("close")
    },

}

if (scriptArgs.length != 2)
    print("usage: mutool run trace-device.js document.pdf pageNumber")

```

(continues on next page)

(continued from previous page)

```

else {
    var doc = new Document(scriptArgs[0]);
    var page = doc.loadPage(parseInt(scriptArgs[1])-1);
    page.run(traceDevice, Identity);
}

```

## Advanced examples

### Create a PDF from rendered page thumbnails

Listing 7: docs/examples/create-thumbnail.js

```

// Create a PDF containing thumbnails of pages rendered from another PDF.

var pdf = new PDFDocument()

var subdoc = new Document("pdfref17.pdf")

var resources = { XObject: {} }

var contents = new Buffer()
for (var i=0; i < 5; ++i) {
    var pixmap = subdoc.loadPage(1140+i).toPixmap([0.2,0,0,0.2,0,0],
↳ DeviceRGB, true)
    resources.XObject["Im" + i] = pdf.addImage(new Image(pixmap))
    contents.writeLine("q 100 0 0 150 " + (50+100*i) + " 50 cm /Im" + i +
↳ " Do Q")
}

var page = pdf.addPage([0,0,100+i*100,250], 0, resources, contents)
pdf.insertPage(-1, page)

pdf.save("out.pdf")

```

A tool for running *JavaScript* programs with access to the *MuPDF* library functions.

## 3.4 mutool JavaScript API

### 3.4.1 Global *MuPDF* methods

#### **print(...)**

Print arguments to stdout, separated by spaces and followed by a newline.

#### **Arguments**

- ... – Arguments to print.

#### **Example**

```
var document = new Document("my_pdf.pdf");
print(document); // [object pdf_document]
```

**write(...)**

Print arguments to stdout, separated by spaces.

**Arguments**

- ... – Arguments to print.

**repr(object)**

Format the object into a string with *JavaScript* syntax.

**Example**

```
var document = new Document("my_pdf.pdf");
repr(document); // "[userdata pdf_document]"
```

**gc(report)**

Run the garbage collector to free up memory. Optionally report statistics on the garbage collection.

**Arguments**

- **report** – Boolean If *true* then the results will be printed to stdout.

**load(fileName)**

Load and execute script in *fileName*.

**Arguments**

- **fileName** – String *JavaScript* file to load.

**read(fileName)**

Read the contents of a file and return them as a *UTF-8* decoded string.

**Arguments**

- **fileName** – String.

**readline()**

Read one line of input from *stdin* and return it as a string.

**Returns**

String.

**require(module)**

Load a *JavaScript* module.

**setUserCSS(userStyleSheet, usePublisherStyles)**

Set user styles and whether to use publisher styles when laying out reflowable documents.

**Arguments**

- **userStyleSheet** – Link to *CSS* stylesheet file.
- **usePublisherStyles** – Boolean.

**quit()**

Exit the shell.



### 3.4.2 Matrices and Rectangles

These are not classes as such but array definitions. All dimensions are in points unless otherwise specified.

#### Matrices

Matrices are simply 6-element arrays representing a 3-by-3 transformation matrix as:

```
/ a b 0 \
| c d 0 |
\ e f 1 /
```

This matrix is represented in *JavaScript* as `[a,b,c,d,e,f]`.

#### Properties

##### Identity

The identity matrix, short hand for `[1,0,0,1,0,0]`.

#### Methods

##### Scale(*sx*, *sy*)

Returns a scaling matrix, short hand for `[sx,0,0,sy,0,0]`.

##### Returns

`[a,b,c,d,e,f]`.

##### Translate(*tx*, *ty*)

Return a translation matrix, short hand for `[1,0,0,1,tx,ty]`.

##### Returns

`[a,b,c,d,e,f]`.

##### Rotate(*theta*)

Return a rotation matrix, short hand for `[cos(theta),sin(theta),-sin(theta),cos(theta),0,0]`.

##### Returns

`[a,b,c,d,e,f]`.

##### Concat(*a*, *b*)

Concatenate matrices *a* and *b*. Bear in mind that matrix multiplication is not commutative.

##### Returns

`[a,b,c,d,e,f]`.

#### Rectangles

Rectangles are 4-element arrays, specifying the minimum and maximum corners (typically upper left and lower right, in a coordinate space with the origin at the top left with descending y): `[ulx,uly,lrx,lry]`.

If the minimum x coordinate is bigger than the maximum x coordinate, *MuPDF* treats the rectangle as infinite in size.

### 3.4.3 Colors

Colors are specified as arrays with the appropriate number of components for the *color space*. Each number is a floating point between 0 and 1 for the color value.

Therefore colors are represented as an array of up to 4 component values.

For example:

- In the DeviceCMYK color space a color would be [Cyan,Magenta,Yellow,Black]. A full magenta color would therefore be [0,1,0,0].
- In the DeviceRGB color space a color would be [Red,Green,Blue]. A full green color would therefore be [0,1,0].
- In the DeviceGray color space a color would be [Black]. A full black color would therefore be [1].

#### Color parameters

This object is a dictionary with keys for:

**renderingIntent**

Either of “Perceptual”, “RelativeColorimetric”, “Saturation” or “AbsoluteColorimetric”.

**blackPointCompensation**

True if black point compensation is activated.

**overPrinting**

True if overprint is activated.

**overPrintMode**

The overprint mode. Can be either 0 or 1.

#### Alpha

Alpha values are floats between 0 and 1, whereby 0 denotes full transparency & 1 denotes full opacity.

### 3.4.4 Object Protocols

The following objects are standard *JavaScript* objects with assumed properties (i.e. they follow their outlined protocol). They are used throughout the *mutool API* to support object types for various methods.

#### Link Dictionary Object

A link is a dictionary with keys for:

**bounds**

A rectangle describing the link’s location on the page.

**uri**

A uri describing either a document internal destination or a *URL* for a web page.

## Link Destination Object

A link destination points to a location within a document and how a document viewer should show that destination.

It consists of a dictionary with keys for:

**chapter**

The chapter within the document.

**page**

The page within the document.

**type**

Either “Fit”, “FitB”, “FitH”, “FitBH”, “FitV”, “FitBV”, “FitR” or “XYZ”, controlling which of the keys below exist.

**x**

The left coordinate, valid for “FitV”, “FitBV”, “FitR” and “XYZ”.

**y**

The top coordinate, valid for “FitH”, “FitBH”, “FitR” and “XYZ”.

**width**

The width of the zoomed in region, valid for “XYZ”.

**height**

The height of the zoomed in region, valid for “XYZ”.

**zoom**

The zoom factor, valid for “XYZ”.

## File Specification Object

This object is used to represent a file.

In order to retrieve information from this object see methods described within *Embedded files in PDFs*.

## Embedded File Object

This Object contains metadata about an embedded file, it has properties for:

**filename**

The name of the embedded file.

**mimetype**

The *MIME* type of the embedded file, or *undefined* if none exists.

**size**

The size in bytes of the embedded file contents.

**creationDate**

The creation date of the embedded file.

**modificationDate**

The modification date of the embedded file.

## PDF Journal Object

This Object contains a numbered array of operations and a reference into this list indicating the current position.

### **position**

The current position in the journal.

### **steps**

An array containing the name of each step in the journal.

## Stroking State Object

The stroking state is a dictionary with keys for:

- **startCap, dashCap, endCap**  
“Butt”, “Round”, “Square”, or “Triangle”.
- **lineCap**  
Set startCap, dashCap, and endCap all at once.
- **lineJoin**  
“Miter”, “Round”, “Bevel”, or “MiterXPS”.
- **lineWidth**  
Thickness of the line.
- **miterLimit**  
Maximum ratio of the miter length to line width, before beveling the join instead.
- **dashPhase**  
Starting offset for dash pattern.
- **dashes**  
Array of on/off dash lengths.

### **Example:**

```
{dashes:[5,10], lineWidth:3, lineCap:'Round'}
```

## Outline Iterator Object

This Object has properties for:

### **title**

The title of the item.

### **uri**

A *URI* pointing to the destination. Likely to be a document internal link that can be resolved by *Document.resolveLink()*, otherwise a link to a web page.

### **open**

*True* if the item should be opened when shown in a tree view.

## Text Layout Object

A description of layouted text value from a text widget with keys:

### **matrix**

Normal transform matrix for the layouted text.

### **invMatrix**

Inverted transform matrix for the layouted text.

### **lines**

An array of text lines belonging to the layouted text, a `lines` object contains:

- `x` The X coordinate for the text line.
- `y` The Y coordinate for the text line.
- `fontSize` The text size used for the layouted text line.
- *Welcome to MuPDF* The index of the beginning of the line in the text string.
- *rect* The bounding rectangle for the text line.
- `chars` An array of characters in the text line.

A `chars` object contains:

- `x` The position of the character.
- `advance` The advance of the character.
- *Welcome to MuPDF* The index of the character in the text string.
- *rect* The bounding *Rectangle* for the character.

## Signature Configuration Object

A signature configuration object has properties with `Boolean` values as follows:

### **showLabels**

Whether to include both labels and values or just values on the right hand side.

### **showDN**

Whether to include the distinguished name on the right hand side.

### **showTextName**

Whether to include the name of the signatory on the right hand side.

### **showDate**

Whether to include the date of signing on the right hand side.

### **showGraphicName**

Whether to include the signatory name on the left hand side.

### **showLogo**

Whether to include the *MuPDF* logo in the background.

## Placement Result Object

### **filled**

The rectangle of the actual area that was used.

### **more**

*True* if more content remains to be placed, otherwise *false* if all content fits in the Story.

## 3.4.5 Buffer

The Buffer objects are used for working with binary data. They can be used much like arrays, but are much more efficient since they only store bytes.

### **new Buffer()**

*Constructor method.*

Create a new empty buffer.

#### **Returns**

Buffer.

#### **Example**

```
var buffer = new Buffer();
```

### **readFile(fileName)**

*Constructor method.*

Create a new buffer with the contents of a file.

#### **Arguments**

- **fileName** – The path to the file to read.

#### **Returns**

Buffer.

#### **Example**

```
var buffer = readFile("my_file.pdf");
```

---

## Instance properties

### **length**

The number of bytes in the buffer. Read-only.

### **[n]**

Read/write the byte at index 'n'. Will throw exceptions on out of bounds accesses.

#### **Example**

```
var byte = buffer[0];
```

---

## Instance methods

**writeByte(*b*)**

Append a single byte to the end of the buffer.

**Arguments**

- **b** – The byte value.

**writeRune(*c*)**

Encode a unicode character as UTF-8 and append to the end of the buffer.

**Arguments**

- **c** – The character value.

**writeLine(...)**

Append arguments to the end of the buffer, separated by spaces, ending with a newline.

**Arguments**

- ... – List of arguments.

**write(...)**

Append arguments to the end of the buffer, separated by spaces.

**Arguments**

- ... – List of arguments.

**writeBuffer(*data*)**

Append the contents of the ‘data’ buffer to the end of the buffer.

**Arguments**

- **data** – Data buffer.

**save(*fileName*)**

Write the contents of the buffer to a file.

**Arguments**

- **fileName** – Filename to save to.

### 3.4.6 Document

*MuPDF* can open many document types (*PDF*, *XPS*, *CBZ*, *EPUB*, *FB2* and a handful of image formats).

**new Document(*fileName*)**

*Constructor method.*

Open the named document.

**Arguments**

- **fileName** – Filename to open.

**Returns**

Document.

**Example**

```
var document = new Document("my_pdf.pdf");
```

### **needsPassword()**

Returns *true* if a password is required to open this password protected PDF.

#### **Returns**

Boolean.

### **authenticatePassword(*password*)**

Returns *true* if the password matches.

#### **Arguments**

- **password** – The password to attempt authentication with.

#### **Returns**

Boolean.

### **hasPermission(*permission*)**

Returns *true* if the document has permission for “print”, “annotate”, “edit” or “copy”.

#### **Arguments**

- **permission** – String The permission to seek for, e.g. “edit”.

#### **Returns**

Boolean.

### **getMetaData(*key*)**

Return various meta data information. The common keys are: `format`, `encryption`, `info:Author`, and `info:Title`.

#### **Arguments**

- **key** – String.

#### **Returns**

String.

### **setMetaData(*key*, *value*)**

Set document meta data information field to a new value.

#### **Arguments**

- **key** – String.
- **value** – String.

### **isReflowable()**

Returns true if the document is reflowable, such as *EPUB*, *FB2* or *XHTML*.

#### **Returns**

Boolean.

### **layout(*pageWidth*, *pageHeight*, *fontSize*)**

Layout a reflowable document (*EPUB*, *FB2*, or *XHTML*) to fit the specified page and font size.

#### **Arguments**

- **pageWidth** – Int.
- **pageHeight** – Int.
- **fontSize** – Int.



**countPages()**

Count the number of pages in the document. This may change if you call the layout function with different parameters.

**Returns**

Int.

**loadPage(*number*)**

Returns a Page (or PDFPage) object for the given page number. Page number zero (0) is the first page in the document.

**Returns**

Page or PDFPage.

**loadOutline()**

Returns an array with the outline (also known as “table of contents” or “bookmarks”). In the array is an object for each heading with the property ‘title’, and a property ‘page’ containing the page number. If the object has a ‘down’ property, it contains an array with all the sub-headings for that entry.

**Returns**

[].

**outlineIterator()**

Returns an *OutlineIterator* for the document outline.

**Returns**

OutlineIterator.

**resolveLink(*uri*)**

Resolve a document internal link *URI* to a link destination.

**Arguments**

- **uri** – String.

**Returns**

*Link destination*.

**formatLinkURI(*linkDestination*)**

Format a document internal link destination object to a *URI* string suitable for *createLink()*.

**Arguments**

- **linkDestination** – *Link destination*.

**Returns**

String.

**isPDF()**

Returns *true* if the document is a *PDF* document.

**Returns**

Boolean.

### 3.4.7 Page

#### **bound()**

Returns a *rectangle* containing the page dimensions.

##### **Returns**

[ulx,uly,lrx,lry].

#### **run(device, transform, skipAnnotations)**

Calls device functions for all the contents on the page, using the specified *transform matrix*. The device can be one of the built-in devices or a *JavaScript* object with methods for the device calls. The *transform* maps from user space points to device space pixels. If *skipAnnotations* is *true* then annotations are ignored.

##### **Arguments**

- **device** – The device object.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **skipAnnotations** – Boolean.

#### **toPixmap(transform, colorspace, alpha, skipAnnotations)**

Render the page into a *Pixmap*, using the *transform* and *colorspace*. If *alpha* is *true*, the page will be drawn on a transparent background, otherwise white.

##### **Arguments**

- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **colorspace** – ColorSpace.
- **alpha** – Boolean.
- **skipAnnotations** – Boolean.

#### **toDisplayList(skipAnnotations)**

Record the contents on the page into a DisplayList.

##### **Arguments**

- **skipAnnotations** – Boolean.

#### **toStructuredText(options)**

Extract the text on the page into a StructuredText object. The options argument is a comma separated list of flags: “preserve-ligatures”, “preserve-whitespace”, “preserve-spans”, and “preserve-images”.

##### **Arguments**

- **options** – String.

##### **Returns**

StructuredText.

#### **search(needle)**

Search for needle text on the page, and return an array with *rectangles* of all matches found.

##### **Arguments**

- **needle** – String.

##### **Returns**

[].

**getLinks()**

Return an array of all the links on the page. Each link is an object with a ‘bounds’ property, and either a ‘page’ or ‘uri’ property, depending on whether it’s an internal or external link. See: [Links](#).

**Returns**

[].

**createLink**(*rect*, *destinationUri*)

Create a new link within the rectangle on the page, linking to the destination URI string.

**Arguments**

- **rect** – [Rectangle](#) for the link.
- **destinationUri** – String.

**Returns**

Object [Link dictionary](#).

**Example**

```
var link = page.createLink([0,0,100,100],"http://mupdf.com");
```

**deleteLink**(*link*)

Delete the link from the page.

**Arguments**

- **link** – Object [Link dictionary](#).

**isPDF()**

Returns *true* if the page is from a *PDF* document.

**Returns**

Boolean.

### 3.4.8 StructuredText

StructuredText objects hold text from a page that has been analyzed and grouped into blocks, lines and spans.

**search**(*needle*)

Search the text for all instances of *needle*, and return an array with [rectangles](#) of all matches found.

**Arguments**

- **needle** – String.

**Returns**

[].

**highlight**(*p*, *q*)

Return an array with [rectangles](#) needed to highlight a selection defined by the start and end points.

**Arguments**

- **p** – Start point in format {x:y}.
- **q** – End point in format {x:y}.

**copy**(*p*, *q*)

Return the text from the selection defined by the start and end points.

**Arguments**

- **p** – Start point in format {*x*:*y*}.
- **q** – End point in format {*x*:*y*}.

### 3.4.9 ColorSpace

**Properties**

DeviceGray

The default grayscale colorspace.

DeviceRGB

The default RGB colorspace.

DeviceBGR

The default RGB colorspace, but with components in reverse order.

DeviceCMYK

The default CMYK colorspace.

---

**Instance methods**

**getNumberOfComponents**()

A grayscale colorspace has one component, RGB has 3, CMYK has 4, and DeviceN may have any number of components.

**Example**

```
var cs = DeviceRGB;
var num = cs.getNumberOfComponents();
print(num); //3
```

### 3.4.10 Pixmap

A Pixmap object contains a color raster image (short for pixel map). The components in a pixel in the Pixmap are all byte values, with the transparency as the last component. A Pixmap also has a location (*x*, *y*) in addition to its size; so that they can easily be used to represent tiles of a page.

**new Pixmap**(*colorspace*, *bounds*, *alpha*)

*Constructor method.*

Create a new pixmap. The pixel data is **not** initialized; and will contain garbage.

**Arguments**

- **colorspace** – Colorspace.
- **bounds** – [*ulx*, *uly*, *lrx*, *lry*] *Rectangle*.
- **alpha** – Boolean.

**Returns**

Pixmap.

**Example**

```
var pixmap = new Pixmap(DeviceRGB, [0,0,100,100], true);
```

**clear(*value*)**

Clear the pixels to the specified value. Pass 255 for white, or undefined for transparent.

**Arguments**

- **value** – Pixel value.

**bound()**

Return the pixmap bounds.

**Returns**

[ulx,uly,lrx,lry] *Rectangle*.

**getWidth()****Returns**

Int The width value.

**getHeight()****Returns**

Int The height value.

**getNumberOfComponents()**

Number of colors; plus one if an alpha channel is present.

**Returns**

Int Number of color components.

**getAlpha()**

*True* if alpha channel is present.

**Returns**

Boolean.

**getStride()**

Number of bytes per row.

**Returns**

Int.

**getColorSpace()**

Returns the ColorSpace for the Pixmap.

**Returns**

ColorSpace.

**getXResolution()**

Returns the x resolution for the Pixmap.

**Returns**

Int Resolution in dots per inch.

### **getYResolution()**

Returns the y resolution for the Pixmap.

#### **Returns**

Int Resolution in dots per inch.

### **getSample(x, y, k)**

Get the value of component **k** at position **x**, **y** (relative to the image origin: 0, 0 is the top left pixel).

#### **Arguments**

- **x** – X co-ordinate.
- **y** – Y co-ordinate.
- **k** – Component.

#### **Returns**

Int.

### **setResolution(xRes, yRes)**

Set **x** & **y** resolution.

#### **Arguments**

- **xRes** – Int X resolution in dots per inch.
- **yRes** – Int Y resolution in dots per inch.

### **saveAsPNG(fileName)**

Save the Pixmap as a *PNG*. Only works for *Gray* and *RGB* images.

#### **Arguments**

- **fileName** – String.

### **invert()**

Invert all pixels. All components are processed, except alpha which is unchanged.

### **invertLuminance()**

Transform all pixels so that luminance of each pixel is inverted, and the chrominance remains as unchanged as possible. All components are processed, except alpha which is unchanged.

### **gamma(gamma)**

Apply gamma correction to Pixmap. All components are processed, except alpha which is unchanged.

Values  $\geq 0.1$  &  $< 1$  = darken,  $> 1$  &  $< 10$  = lighten.

#### **Arguments**

- **gamma** – Float.

### **tint(black, white)**

Tint all pixels in a *RGB*, *BGR* or *Gray* Pixmap. Map black and white respectively to the given hex *RGB* values.

#### **Arguments**

- **black** – Integer.
- **white** – Integer.

**warp**(*points*, *width*, *height*)

Return a warped subsection of the `Pixmap`, where the result has the requested dimensions.

#### Arguments

- **points** – [`x0`, `y0`, `x1`, `y1`, `x2`, `y2`, `x3`, `y3`, ...] Points give the corner points of a convex quadrilateral within the `Pixmap` to be warped.
- **width** – `Int`.
- **height** – `Int`.

### 3.4.11 DrawDevice

The `DrawDevice` can be used to render to a *Pixmap*; either by running a *Page* with it or by calling its methods directly.

**new DrawDevice**(*transform*, *pixmap*)

*Constructor method.*

Create a device for drawing into a `Pixmap`. The `Pixmap` bounds used should match the transformed page bounds, or you can adjust them to only draw a part of the page.

#### Arguments

- **transform** – [`a`, `b`, `c`, `d`, `e`, `f`]. The transform *matrix*.
- **pixmap** – `Pixmap`.

#### Example

```
var pixmap = new DrawDevice(Identity, pixmap);
```

### 3.4.12 DisplayList

A display list records all the device calls for playback later. If you want to run a page through several devices, or run it multiple times for any other reason, recording the page to a display list and replaying the display list may be a performance gain since then you can avoid reinterpreting the page each time. Be aware though, that a display list will keep all the graphics required in memory, so will increase the amount of memory required.

**new DisplayList**(*mediabox*)

*Constructor method.*

Create an empty display list. The mediabox rectangle has the bounds of the page in points.

#### Arguments

- **mediabox** – [`ulx`, `uly`, `lrx`, `lry`] *Rectangle*.

#### Returns

`DisplayList`.

#### Example

```
var displayList = new DisplayList([0,0,100,100]);
```

**run**(*device*, *transform*)

Play back the recorded device calls onto the device.

#### Arguments

- **device** – Device.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**bound()**

Returns a rectangle containing the dimensions of the display list contents.

**Returns**

[ulx, uly, lrx, lry] *Rectangle*.

**toPixmap(transform, colorspace, alpha)**

Render display list to a Pixmap.

**Arguments**

- **transform** – [a, b, c, d, e, f]. The transform *matrix*.
- **colorspace** – ColorSpace.
- **alpha** – Boolean. If alpha is *true*, the annotation will be drawn on a transparent background, otherwise white.

**toStructuredText(options)**

Extract the text on the page into a StructuredText object. The options argument is a comma separated list of flags: “preserve-ligatures”, “preserve-whitespace”, “preserve-spans”, and “preserve-images”.

**Arguments**

- **options** – String.

**Returns**

StructuredText.

**search(needle)**

Search the display list text for all instances of ‘needle’, and return an array with *rectangles* of all matches found.

**Arguments**

- **needle** – String.

**Returns**

[].

### 3.4.13 DisplayListDevice

**new DisplayListDevice(displayList)**

*Constructor method.*

Create a device for recording onto a display list.

**Arguments**

- **displayList** – DisplayList.

**Returns**

DisplayListDevice.

**Example**

```
var displayListDevice = new DisplayListDevice(my_display_list);
```



### 3.4.14 Device

All built-in devices have the methods listed below. Any function that accepts a device will also accept a *JavaScript* object with the same methods. Any missing methods are simply ignored, so you only need to create methods for the device calls you care about.

Many of the methods take graphics objects as arguments: *Path*, *Text*, *Image* and *Shade*.

Colors are specified as arrays with the appropriate number of components for the color space.

The methods that clip graphics must be balanced with a corresponding *popClip*.

**fillPath**(*path*, *evenOdd*, *transform*, *colorspace*, *color*, *alpha*, *colorParams*)

Fill a path.

#### Arguments

- **path** – Path object.
- **evenOdd** – The *even odd* rule to use.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **colorspace** – The *ColorSpace*.
- **color** – The *color value*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

#### Example

```
device.fillPath(path, false, Identity, DeviceRGB, [1,0,0], 1);
```

**strokePath**(*path*, *stroke*, *transform*, *colorspace*, *color*, *alpha*, *colorParams*)

Stroke a path.

#### Arguments

- **path** – Path object.
- **stroke** – The *stroke dictionary*.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **colorspace** – The *ColorSpace*.
- **color** – The *color value*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

#### Example

```
device.strokePath(path, {dashes:[5,10], lineWidth:3, lineCap:'Round'}, Identity, ↵
↵DeviceRGB, [0,1,0], 0.5);
```

**clipPath**(*path*, *evenOdd*, *transform*)

Clip a path.

#### Arguments

- **path** – Path object.

- **evenOdd** – The *even odd rule* to use.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**clipStrokePath**(*path*, *stroke*, *transform*)

Clip & stroke a path.

**Arguments**

- **path** – Path object.
- **stroke** – The *stroke dictionary*.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**fillText**(*text*, *transform*, *colorspace*, *color*, *alpha*, *colorParams*)

Fill a text object.

**Arguments**

- **text** – Text object.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.
- **colorspace** – The *ColorSpace*.
- **color** – The *color value*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

**strokeText**(*text*, *stroke*, *transform*, *colorspace*, *color*, *alpha*, *colorParams*)

Stroke a text object.

**Arguments**

- **text** – Text object.
- **stroke** – The *stroke dictionary*.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.
- **colorspace** – The *ColorSpace*.
- **color** – The *color value*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

**clipText**(*text*, *transform*)

Clip a text object.

**Arguments**

- **text** – Text object.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**clipStrokeText**(*text*, *stroke*, *transform*)

Clip & stroke a text object.

**Arguments**

- **text** – Text object.
- **stroke** – The *stroke dictionary*.

- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**ignoreText**(text, transform)

Invisible text that can be searched but should not be visible, such as for overlaying a scanned OCR image.

#### Arguments

- **text** – Text object.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**fillShade**(shade, transform, alpha, colorParams)

Fill a shade (a.k.a. gradient).

---

**Note:** The details of gradient fills are not exposed to *JavaScript* yet.

---

#### Arguments

- **shade** – The gradient.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

**fillImage**(image, transform, alpha, colorParams)

Draw an image. An image always fills a unit rectangle [0, 0, 1, 1], so must be transformed to be placed and drawn at the appropriate size.

#### Arguments

- **image** – Image object.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

**fillImageMask**(image, transform, colorspace, color, alpha, colorParams)

An image mask is an image without color. Fill with the color where the image is opaque.

#### Arguments

- **image** – Image object.
- **transform** – [a, b, c, d, e, f]. The transform *matrix*.
- **colorspace** – The *ColorSpace*.
- **color** – The *color value*.
- **alpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

**clipImageMask**(image, transform)

Clip graphics using the image to mask the areas to be drawn.

#### Arguments

- **image** – Image object.

- **transform** – [a, b, c, d, e, f]. The transform *matrix*.

**popClip()**

Pop the clip mask installed by the last clipping operation.

**beginMask**(*area, luminosity, backdropColorspace, backdropColor, backdropAlpha, colorParams*)

Create a soft mask. Any drawing commands between *beginMask* and *endMask* are grouped and used as a clip mask.

**Arguments**

- **area** – Path Mask area.
- **luminosity** – Boolean If luminosity is *true*, the mask is derived from the luminosity (grayscale value) of the graphics drawn; otherwise the color is ignored completely and the mask is derived from the alpha of the group.
- **backdropColorspace** – The *ColorSpace*.
- **backdropColor** – The *color value*.
- **backdropAlpha** – The *alpha value*.
- **colorParams** – The *color parameters object*.

**endMask()**

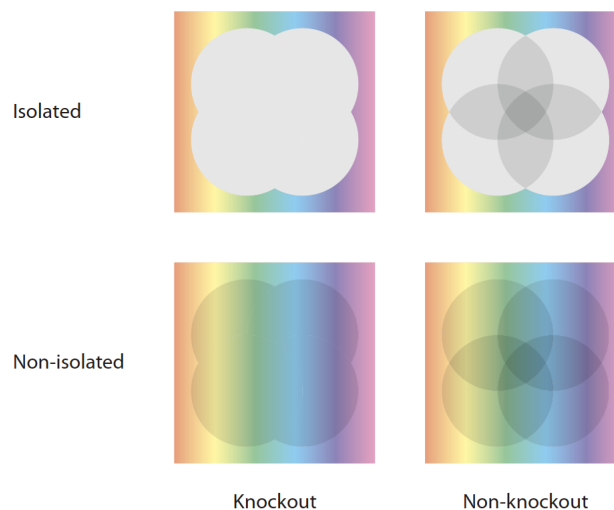
Ends the mask.

**beginGroup**(*area, isolated, knockout, blendmode, alpha*)

Push/pop a transparency blending group. See the PDF reference for details on *isolated* and *knockout*.

**Arguments**

- **area** – Path Blend area.
- **isolated** – Boolean.
- **knockout** – Boolean.
- **blendmode** – Blendmode is one of the standard *PDF* blend modes: “Normal”, “Multiply”, “Screen”, etc.
- **alpha** – The *alpha value*.



**endGroup()**

Ends the blending group.

**beginTile**(*areaRect*, *viewRect*, *xStep*, *yStep*, *transform*, *id*)

Draw a tiling pattern. Any drawing commands between *beginTile* and *endTile* are grouped and then repeated across the whole page. Apply a clip mask to restrict the pattern to the desired shape.

**Arguments**

- **areaRect** – [*ulx*, *uly*, *lrx*, *lry*] *Rectangle*.
- **viewRect** – [*ulx*, *uly*, *lrx*, *lry*] *Rectangle*.
- **xStep** – Integer representing x step.
- **yStep** – Integer representing y step.
- **transform** – [*a*, *b*, *c*, *d*, *e*, *f*]. The transform *matrix*.
- **id** – Integer The purpose of *id* is to allow for efficient caching of rendered tiles. If *id* is 0, then no caching is performed. If it is non-zero, then it assumed to uniquely identify this tile.

**endTile()**

Ends the tiling pattern.

**close()**

Tell the device that we are done, and flush any pending output.

**beginLayer**(*tag*)

Begin a marked-content layer with the given tag.

**Arguments**

- **tag** – String.

**endLayer()**

End a marked-content layer.

### 3.4.15 Path

A Path object represents vector graphics as drawn by a pen. A path can be either stroked or filled, or used as a clip mask.

**new Path()**

*Constructor method.*

Create a new empty path.

**Returns**

Path.

**Example**

```
var path = new Path();
```

**bound**(*stroke*, *transform*)

Return a bounding rectangle for the path.

**Arguments**

- **stroke** – Float The stroke for the path.
- **transform** – [a, b, c, d, e, f]. The transform *matrix* for the path.

**Returns**

[ulx, uly, lrx, lry] *Rectangle*.

**moveTo**(x, y)

Lift and move the pen to the coordinate.

**Arguments**

- **x** – X coordinate.
- **y** – Y coordinate.

**lineTo**(x, y)

Draw a line to the coordinate.

**Arguments**

- **x** – X coordinate.
- **y** – Y coordinate.

**curveTo**(x1, y1, x2, y2, x3, y3)

Draw a cubic bezier curve to (x3, y3) using (x1, y1) and (x2, y2) as control points.

**Arguments**

- **x1** – X1 coordinate.
- **y1** – Y1 coordinate.
- **x2** – X2 coordinate.
- **y2** – Y2 coordinate.
- **x3** – X3 coordinate.
- **y3** – Y3 coordinate.

**curveToV**(cx, cy, ex, ey)

Draw a cubic bezier curve to (ex, ey) using the start point and (cx, cy) as control points.

**Arguments**

- **cx** – CX coordinate.
- **cy** – CY coordinate.
- **ex** – EX coordinate.
- **ey** – EY coordinate.

**curveToY**(cx, cy, ex, ey)

Draw a cubic bezier curve to (ex, ey) using the (cx, cy) and (ex, ey) as control points.

**Arguments**

- **cx** – CX coordinate.
- **cy** – CY coordinate.
- **ex** – EX coordinate.
- **ey** – EY coordinate.

**closePath()**

Close the path by drawing a line to the last *moveTo*.

**rect(x1, y1, x2, y2)**

Shorthand for *moveTo*, *lineTo*, *lineTo*, *lineTo*, *closePath* to draw a rectangle.

**Arguments**

- **x1** – X1 coordinate.
- **y1** – Y1 coordinate.
- **x2** – X2 coordinate.
- **y2** – Y2 coordinate.

**walk(pathWalker)**

Call *moveTo*, *lineTo*, *curveTo* and *closePath* methods on the pathWalker object to replay the path.

**Arguments**

- **pathWalker** – The path walker object. A user definable *JavaScript* object which can be used to trigger your own functions on the path methods.

---

**Note:** A path walker object has callback methods that are called when *walk()* walks over *moveTo*, *lineTo*, *curveTo* and *closePath* operators in a Path.

---

**Example**

```
var myPathWalker = {
  moveTo: function (x, y) { ... do whatever ... },
 .lineTo: function (x, y) { ... do whatever ... },
}

path.walk(myPathWalker);
```

**transform(transform)**

Transform path by the given transform matrix.

**Arguments**

- **transform** – [a, b, c, d, e, f]. The transform *matrix* for the path.

### 3.4.16 Text

A Text object contains text.

**new Text()**

*Constructor method.*

Create a new empty text object.

**Returns**

Text.

**Example**

```
var text = new Text();
```

**showGlyph**(font, transform, glyph, unicode, wmode)

Add a glyph to the text object.

Transform is the text matrix, specifying font size and glyph location. For example: [size,0,0,-size,x,y].

Glyph and unicode may be -1 for n-to-m cluster mappings. For example, the “fi” ligature would be added in two steps: first the glyph for the ‘fi’ ligature and the unicode value for ‘f’; then glyph -1 and the unicode value for ‘i’.

**Arguments**

- **font** – Font object.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **glyph** –
- **unicode** –
- **wmode** – 0 for horizontal writing, and 1 for vertical writing.

**showString**(font, transform, string)

Add a simple string to the Text object. Will do font substitution if the font does not have all the unicode characters required.

**Arguments**

- **font** – Font object.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **string** – String content for Text object.

**walk**(textWalker)

Call the *showGlyph* method on the textWalker object for each glyph in the text object.

**Arguments**

- **textWalker** – The text walker object. A user definable *JavaScript* object which can be used to trigger your own functions on the text methods.

**Example**

```
var myTextWalker = {
  showGlyph: function (font, transform, glyph, unicode, wmode) { ... do whatever .
  ↪ .. },
}

text.walk(myTextWalker);
```

### 3.4.17 Font

Font objects can be created from *TrueType*, *OpenType*, *Type1* or *CFF* fonts. In *PDF* there are also special *Type3* fonts.

**new Font**(ref)

*Constructor method.*

Create a new font, either using a built-in font name or a file name.

The built-in standard *PDF* fonts are:

- *Times-Roman*.
- *Times-Italic*.



- *Times-Bold*.
- *Times-BoldItalic*.
- *Helvetica*.
- *Helvetica-Oblique*.
- *Helvetica-Bold*.
- *Helvetica-BoldOblique*.
- *Courier*.
- *Courier-Oblique*.
- *Courier-Bold*.
- *Courier-BoldOblique*.
- *Symbol*.
- *ZapfDingbats*.

The built-in CJK fonts are referenced by language code: zh-Hant, zh-Hans, ja, ko.

#### Arguments

- **ref** – Font name or file name.

#### Returns

Font.

#### Example

```
var font = new Font("Times-Roman");
```

#### getName()

Get the font name.

#### Returns

String.

#### encodeCharacter(*unicode*)

Get the glyph index for a unicode character. Glyph zero (.notdef) is returned if the font does not have a glyph for the character.

#### Arguments

- **unicode** – The unicode character.

#### Returns

Glyph index.

#### advanceGlyph(*glyph*, *wmode*)

Return advance width for a glyph in either horizontal or vertical writing mode.

#### Arguments

- **glyph** – The glyph.
- **wmode** – 0 for horizontal writing, and 1 for vertical writing.

#### Returns

Width for a glyph.

### **isBold()**

Returns *true* if font is bold.

#### **Returns**

Boolean.

### **isItalic()**

Returns *true* if font is italic.

#### **Returns**

Boolean.

### **isMono()**

Returns *true* if font is monospaced.

#### **Returns**

Boolean.

### **isSerif()**

Returns *true* if font is serif.

#### **Returns**

Boolean.

## **3.4.18 Image**

Image objects are similar to `Pixmap`s, but can contain compressed data.

### **new Image(ref)**

*Constructor method.*

Create a new image from a `Pixmap` data, or load an image from a file.

#### **Returns**

Image.

#### **Example**

```
var image = new Image("my-image-file.png");
```

### **getWidth()**

Get the image width in pixels.

#### **Returns**

The width value.

### **getHeight()**

Get the image height in pixels.

#### **Returns**

The height value.

### **getXResolution()**

Returns the x resolution for the Image.

#### **Returns**

Int Image resolution in dots per inch.

**getYResolution()**

Returns the y resolution for the Image.

**Returns**

Int Image resolution in dots per inch.

**getColorSpace()**

Returns the ColorSpace for the Image.

**Returns**

ColorSpace.

**getNumberOfComponents()**

Number of colors; plus one if an alpha channel is present.

**Returns**

Integer.

**getBitsPerComponent()**

Returns the number of bits per component.

**Returns**

Integer.

**getInterpolate()**

Returns *true* if interpolated was used during decoding.

**Returns**

Boolean.

**getImageMask()**

Returns *true* if this image is an image mask.

**Returns**

Boolean.

**getMask()**

Get another Image used as a mask for this one.

**Returns**

Image (or null).

**toPixmap(*scaledWidth*, *scaledHeight*)**

Create a Pixmap from the image. The *scaledWidth* and *scaledHeight* arguments are optional, but may be used to decode a down-scaled Pixmap.

**Arguments**

- **scaledWidth** – Width value.
- **scaledHeight** – Height value.

**Returns**

Pixmap.

### 3.4.19 DocumentWriter

DocumentWriter objects are used to create new documents in several formats.

**new DocumentWriter**(*filename, format, options*)

*Constructor method.*

Create a new document writer to create a document with the specified format and output options. If format is null it is inferred from the filename extension. The options argument is a comma separated list of flags and key-value pairs.

The output format & options are the same as in the *mutool convert* command.

#### Arguments

- **filename** – The file name to output to.
- **format** – The file format.
- **options** – The options as key-value pairs.

#### Returns

DocumentWriter.

#### Example

```
var writer = new DocumentWriter("out.pdf", "PDF", "");
```

**beginPage**(*mediabox*)

Begin rendering a new page. Returns a Device that can be used to render the page graphics.

#### Arguments

- **mediabox** – [ulx,uly,lrx,lry] *Rectangle*.

#### Returns

Device.

**endPage**(*device*)

Finish the page rendering. The argument must be the same Device object that was returned by the *beginPage* method.

#### Arguments

- **mediabox** – Device.

**close**()

Finish the document and flush any pending output.

### 3.4.20 PDFDocument

With *MuPDF* it is also possible to create, edit and manipulate *PDF* documents using low level access to the objects and streams contained in a *PDF* file. A PDFDocument object is also a Document object. You can test a Document object to see if it is safe to use as a PDFDocument by calling `document.isPDF()`.

**new PDFDocument**()

*Constructor method.*

Create a new empty *PDF* document.

#### Returns

PDFDocument.

**Example**

```
var pdfDocument = new PDFDocument();
```

**new PDFDocument(fileName)**

Constructor method.

Load a *PDF* document from file.

**Returns**

PDFDocument.

**Example**

```
var pdfDocument = new PDFDocument("my-file.pdf");
```

**save(fileName, options)**

Write the PDFDocument to file. The write options are a string of comma separated options (see the *mutool convert options*).

**Arguments**

- **fileName** – The name of the file to save to.

**Options**

The options as key-value pairs.

**canBeSavedIncrementally()**

Returns *true* if the document can be saved incrementally, e.g. repaired documents or applying redactions prevents incremental saves.

**Returns**

Boolean.

**countVersions()**

Returns the number of versions of the document in a *PDF* file, typically 1 + the number of updates.

**Returns**

Integer.

**countUnsavedVersions()**

Returns the number of unsaved updates to the document.

**Returns**

Integer.

**validateChangeHistory()**

Check the history of the document, return the last version that checks out OK. Returns 0 if the entire history is OK, 1 if the next to last version is OK, but the last version has issues, etc.

**Returns**

Integer.

**hasUnsavedChanges()**

Returns *true* if the document has been saved since it was last opened or saved.

**Returns**

Boolean.

**wasPureXFA()**

Returns *true* if the document was an *XFA* form without *AcroForm* fields.

**Returns**

Boolean.

**wasRepaired()**

Returns *true* if the document was repaired when opened.

**Returns**

Boolean.

**getTrailer()**

The trailer dictionary. This contains indirect references to the “Root” and “Info” dictionaries. See: [PDF object access](#).

**Returns**

The trailer dictionary.

**countObjects()**

Return the number of objects in the *PDF*. Object number 0 is reserved, and may not be used for anything. See: [PDF object access](#).

**Returns**

Object count.

**createObject()**

Allocate a new numbered object in the *PDF*, and return an indirect reference to it. The object itself is uninitialized.

**Returns**

The new object.

**deleteObject(obj)**

Delete the object referred to by the indirect reference.

**Arguments**

- **obj** – The object to delete.

---

**PDF JavaScript actions**

**enableJS()**

Enable interpretation of document *JavaScript* actions.

**disableJS()**

Disable interpretation of document *JavaScript* actions.

**isJSSupported()**

Returns *true* if interpretation of document *JavaScript* actions is supported.

**Returns**

Boolean.

**setJSEventListener(listener)**

Calls the listener whenever a document *JavaScript* action triggers an event.

**Arguments**

- **listener** – {} The *JavaScript* listener function.

---

**Note:** At present this listener will only trigger when a document *JavaScript* action triggers an alert.

---

### PDF journalling

#### **enableJournal()**

Activate journalling for the document.

#### **getJournal()**

Returns a *PDF Journal Object*.

##### **Returns**

Object *PDF Journal Object*.

#### **beginOperation()**

Begin a journal operation

#### **beginImplicitOperation()**

Begin an implicit journal operation. Implicit operations are operations that happen due to other operations, e.g. updating an annotation.

#### **endOperation()**

End a previously started normal or implicit operation. After this it can be undone/redone using the methods below.

#### **canUndo()**

Returns *true* if undo is possible in this state.

##### **Returns**

Boolean.

#### **canRedo()**

Returns *true* if redo is possible in this state.

##### **Returns**

Boolean.

#### **undo()**

Move backwards in the undo history. Changes to the document after this throws away all subsequent history.

#### **redo()**

Move forwards in the undo history.

---

### PDF Object Access

A *PDF* document contains objects, similar to those in *JavaScript*: arrays, dictionaries, strings, booleans, and numbers. At the root of the *PDF* document is the trailer object; which contains pointers to the meta data dictionary and the catalog object which contains the pages and other information.

Pointers in *PDF* are also called indirect references, and are of the form “32 0 R” (where 32 is the object number, 0 is the generation, and R is magic syntax). All functions in *MuPDF* dereference indirect references automatically.

*PDF* has two types of strings: */Names* and *(Strings)*. All dictionary keys are names.

Some dictionaries in *PDF* also have attached binary data. These are called streams, and may be compressed.

---

**Note:** PDFObjects are always bound to the document that created them. Do **NOT** mix and match objects from one document with another document!

---

### **addObject(*obj*)**

Add ‘obj’ to the *PDF* as a numbered object, and return an indirect reference to it.

#### **Arguments**

- **obj** – Object to add.

### **addStream(*buffer*, *object*)**

Create a stream object with the contents of **buffer**, add it to the *PDF*, and return an indirect reference to it. If **object** is defined, it will be used as the stream object dictionary.

#### **Arguments**

- **buffer** – Buffer object.
- **object** – The object to stream to.

### **addRawStream(*buffer*, *object*)**

Create a stream object with the contents of **buffer**, add it to the *PDF*, and return an indirect reference to it. If **object** is defined, it will be used as the stream object dictionary. The **buffer** must contain already compressed data that matches the “Filter” and “DecodeParms”.

#### **Arguments**

- **buffer** – Buffer object.
- **object** – The object to stream to.

### **newNull()**

Create a new null object.

#### **Returns**

PDFObject.

### **newBoolean(*boolean*)**

Create a new boolean object.

#### **Arguments**

- **boolean** – The boolean value.

#### **Returns**

PDFObject.

### **newInteger(*number*)**

Create a new integer object.

#### **Arguments**

- **number** – The number value.

#### **Returns**

PDFObject.



**newReal**(*number*)

Create a new real number object.

**Arguments**

- **number** – The number value.

**Returns**

PDFObject.

**newString**(*string*)

Create a new string object.

**Arguments**

- **string** – String.

**Returns**

PDFObject.

**newByteString**(*byteString*)

Create a new byte string object.

**Arguments**

- **byteString** – String.

**Returns**

PDFObject.

**newName**(*string*)

Create a new name object.

**Arguments**

- **string** – The string value.

**Returns**

PDFObject.

**newIndirect**(*objectNumber*, *generation*)

Create a new indirect object.

**Arguments**

- **objectNumber** – The string value.
- **generation** – The string value.

**Returns**

PDFObject.

**newArray**()

Create a new array object.

**Returns**

PDFObject.

**newDictionary**()

Create a new dictionary object.

**Returns**

PDFObject.

## PDF Page Access

All page objects are structured into a page tree, which defines the order the pages appear in.

### **countPages()**

Number of pages in the document.

#### **Returns**

Page number.

### **findPage(*number*)**

Return the PDFPage object for a page number.

#### **Arguments**

- **number** – The page number, the first page is number zero.

#### **Returns**

PDFPage.

### **findPageNumber(*page*)**

Given a PDFPage object, find the page number in the document.

#### **Returns**

Integer.

### **deletePage(*number*)**

Delete the numbered PDFPage.

#### **Arguments**

- **number** – The page number, the first page is number zero.

### **insertPage(*at*, *page*)**

Insert the PDFPage object in the page tree at the location. If 'at' is -1, at the end of the document.

Pages consist of a content stream, and a resource dictionary containing all of the fonts and images used.

#### **Arguments**

- **at** – The index to insert at.
- **page** – The PDFPage to insert.

### **addPage(*mediabox*, *rotate*, *resources*, *contents*)**

Create a new page object. Note: this function does NOT add it to the page tree.

#### **Arguments**

- **mediabox** – [*ulx*, *uly*, *lrx*, *lry*] *Rectangle*.
- **rotate** – Rotation value.
- **resources** – Resources object.
- **contents** – Contents string. This represents the page content stream - see section 3.7.1 in the PDF 1.7 specification.

#### **Returns**

PDFPage.

### **Example**

Listing 8: docs/examples/pdf-create.js

```
// Create a PDF from scratch using helper functions.

// This example creates a new PDF file from scratch, using helper
// functions to create resources and page objects.
// This assumes a basic working knowledge of the PDF file format.

// Create a new empty document with no pages.
var pdf = new PDFDocument()

// Load built-in font and create WinAnsi encoded simple font resource.
var font = pdf.addSimpleFont(new Font("Times-Roman"))

// Load PNG file and create image resource.
var image = pdf.addImage(new Image("example.png"))

// Create resource dictionary.
var resources = pdf.addObject({
    Font: { Tm: font },
   XObject: { Im0: image },
})

// Create content stream data.
var contents =
    "10 10 280 330 re s\n" +
    "q 200 0 0 200 50 100 cm /Im0 Do Q\n" +
    "BT /Tm 16 Tf 50 50 TD (Hello, world!) Tj ET\n"

// Create a new page object.
var page = pdf.addPage([0,0,300,350], 0, resources, contents)

// Insert page object at the end of the document.
pdf.insertPage(-1, page)

// Save the document to file.
pdf.save("out.pdf", "pretty,ascii,compress-images,compress-fonts")
```

**addSimpleFont(font, encoding)**

Create a PDF object from the Font object as a simple font.

**Arguments**

- **font** – Font.
- **encoding** – The encoding to use. Encoding is either “Latin” (CP-1252), “Greek” (ISO-8859-7), or “Cyrillic” (KOI-8U). The default is “Latin”.

**addCJKFont(font, language, wmode, style)**

Create a PDF object from the Font object as a UTF-16 encoded CID font for the given language (“zh-Hant”, “zh-Hans”, “ko”, or “ja”), writing mode (“H” or “V”), and style (“serif” or “sans-serif”).

**Arguments**

- **font** – Font.

- **language** – String.
- **wmode** – 0 for horizontal writing, and 1 for vertical writing.
- **style** – String.

**addFont**(*font*)

Create a *PDF* object from the *Font* object as an Identity-H encoded CID font.

**Arguments**

- **font** – *Font*.

**addImage**(*image*)

Create a *PDF* object from the *Image* object.

**Arguments**

- **image** – *Image*.

**loadImage**(*obj*)

Load an *Image* from a *PDF* object (typically an indirect reference to an image resource).

**Arguments**

- **obj** – *PDFObject*.

---

The following functions can be used to copy objects from one document to another:

**graftObject**(*object*)

Deep copy an object into the destination document. This function will not remember previously copied objects. If you are copying several objects from the same source document using multiple calls, you should use a graft map instead.

**Arguments**

- **object** – Object to graft.

**graftPage**(*dstDoc*, *dstPageNumber*, *srcDoc*, *srcPageNumber*)

Graft a page and its resources at the given page number from the source document to the requested page number in the destination document.

**Arguments**

- **dstDoc** – Destination document.
- **dstPageNumber** – Destination page number.
- **srcDoc** – Source document.
- **srcPageNumber** – Source page number.

**newGraftMap**()

Create a graft map on the destination document, so that objects that have already been copied can be found again. Each graft map should only be used with one source document. Make sure to create a new graft map for each source document used.

**Returns**

*PDFGraftMap*.

---

**Embedded files in PDFs**

**addEmbeddedFile**(*filename, mimetype, contents, creationDate, modificationDate, addChecksum*)

Embedded a file into the document. If a checksum is added then the file contents can be verified later. An indirect reference to a *File Specification Object* is returned.

#### Arguments

- **filename** – String.
- **mimetype** – String See: [Mimetype](#).
- **contents** – Contents string. This represents the page content stream - see section 3.7.1 in the PDF 1.7 specification.
- **creationDate** – Integer Milliseconds value.
- **modificationDate** – Integer Milliseconds value.
- **addChecksum** – Boolean.

#### Returns

Object *File Specification Object*.

---

**Note:** After embedding a file into a *PDF*, it can be connected to an annotation using *PDFAnnotation.setFilespec()*.

---

**getEmbeddedFileParams**(*fileSpecObject*)

Return an object describing the file referenced by the *fileSpecObject*.

#### Arguments

- **fileSpecObject** – Object *File Specification Object*.

#### Returns

Object *Embedded File Object*.

**getEmbeddedFileContents**(*fileSpecObject*)

Returns a Buffer with the contents of the embedded file referenced by the *fileSpecObject*.

#### Arguments

- **fileSpecObject** – Object *File Specification Object*.

#### Returns

*Buffer*.

**verifyEmbeddedFileChecksum**(*fileSpecObject*)

Verify the *MD5* checksum of the embedded file contents.

**arg fileSpecObject**

Object *File Specification Object*.

**return**

Boolean.

### 3.4.21 PDFGraftMap

#### **graftObject**(*object*)

Use the graft map to copy objects, with the ability to remember previously copied objects.

##### **Arguments**

- **object** – Object to graft.

#### **graftPage**(*map*, *dstPageNumber*, *srcDoc*, *srcPageNumber*)

Graft a page at the given page number from the source document to the requested page number in the destination document connected to the map.

##### **Arguments**

- **map** – PDFGraftMap.
- **dstPageNumber** – Destination page number.
- **srcDoc** – Source document.
- **srcPageNumber** – Source page number.

### 3.4.22 PDFObject

All functions that take PDFObjects, do automatic translation between *JavaScript* objects and PDFObjects using a few basic rules:

- Null, booleans, and numbers are translated directly.
- *JavaScript* strings are translated to *PDF* names, unless they are surrounded by parentheses: “Foo” becomes the *PDF* name /Foo and “(Foo)” becomes the *PDF* string (Foo).
- Arrays and dictionaries are recursively translated to *PDF* arrays and dictionaries. Be aware of cycles though! The translation does NOT cope with cyclic references!
- The translation goes both ways: *PDF* dictionaries and arrays can be accessed similarly to *JavaScript* objects and arrays by getting and setting their properties.

---

#### **Instance properties**

##### **length**

Length of the array.

---

#### **Instance methods**

##### **get**(*ref*)

Access dictionaries and arrays in the PDFObject.

##### **Arguments**

- **ref** – Key or index.

##### **Returns**

The value for the key or index.

**put**(*ref*, *value*)

Put information into dictionaries and arrays in the PDFObject. Dictionaries and arrays can also be accessed using normal property syntax: `obj.Foo = 42`; `delete obj.Foo`; `x = obj[5]`.

**Arguments**

- **ref** – Key or index.
- **value** – The value for the key or index.

**delete**(*ref*)

Delete a reference from a PDFObject.

**Arguments**

- **ref** – Key or index.

**resolve**()

If the object is an indirect reference, return the object it points to; otherwise return the object itself.

**Returns**

Object.

**isArray**()**Returns**

Boolean.

**isDictionary**()**Returns**

Boolean.

**forEach**(*fun*)

Iterate over all the entries in a dictionary or array and call `fun` for each key-value pair.

**Arguments**

- **fun** – Function in the format `function(key, value){...}`.

**push**(*item*)

Append *item* to the end of the array.

**Arguments**

- **item** – Item to add.

**toString**()

Returns the object as a pretty-printed string.

**Returns**

String.

**valueOf**()

Convert primitive *PDF* objects to a corresponding primitive `Null`, `Boolean`, `Number` or `String` *JavaScript* objects. Indirect *PDF* objects get converted to the string “R” while *PDF* names are converted to plain strings. *PDF* arrays or dictionaries are returned unchanged.

**Returns**

`Null` | `Boolean` | `Number` | `String`.

## PDF streams

The only way to access a stream is via an indirect object, since all streams are numbered objects.

### **isIndirect()**

Is the object an indirect reference.

#### **Returns**

Boolean.

### **asIndirect()**

Return the object number the indirect reference points to.

#### **Returns**

Boolean.

### **isStream()**

*True* if the object is an indirect reference pointing to a stream.

#### **Returns**

Boolean.

### **readStream()**

Read the contents of the stream object into a Buffer.

#### **Returns**

Buffer.

### **readRawStream()**

Read the raw, uncompressed, contents of the stream object into a Buffer.

#### **Returns**

Buffer.

### **writeObject(obj)**

Update the object the indirect reference points to.

#### **Arguments**

- **obj** – Object to update.

### **writeStream(buffer)**

Update the contents of the stream the indirect reference points to. This will update the “Length”, “Filter” and “DecodeParms” automatically.

#### **Arguments**

- **buffer** – Buffer.

### **writeRawStream(buffer)**

Update the contents of the stream the indirect reference points to. The buffer must contain already compressed data that matches the “Filter” and “DecodeParms”. This will update the “Length” automatically, but leave the “Filter” and “DecodeParms” untouched.

#### **Arguments**

- **buffer** – Buffer.

---

Primitive *PDF* objects such as booleans, names, and numbers can usually be treated like *JavaScript* values. When that is not sufficient use these functions:



**isNull()**

Returns *true* if the object is a null object.

**Returns**

Boolean.

**isBoolean()**

Returns *true* if the object is a Boolean object.

**Returns**

Boolean.

**asBoolean()**

Get the boolean primitive value.

**Returns**

Boolean.

**isNumber()**

Returns *true* if the object is a Number object.

**Returns**

Boolean.

**asNumber()**

Get the number primitive value.

**Returns**

Integer.

**isName()**

Returns *true* if the object is a Name object.

**Returns**

Boolean.

**asName()**

Get the name as a string.

**Returns**

String.

**isString()**

Returns *true* if the object is a String object.

**Returns**

Boolean.

**asString()**

Convert a “text string” to a *JavaScript* unicode string.

**Returns**

String.

**asByteString()**

Convert a string to an array of byte values.

**Returns**

[...].

### 3.4.23 PDFPage

#### **getAnnotations()**

Return array of all annotations on the page.

##### **Returns**

[].

#### **createAnnotation(*type*)**

Create a new blank annotation of a given *type*.

##### **Arguments**

- **type** – String representing *annotation type*.

##### **Returns**

PDFAnnotation.

#### **Annotation types**

---

**Note:** Annotation types are also referred to as “subtypes”.

---

Name
Text
Link
FreeText
Square
Circle
Polygon
PolyLine
Highlight
Underline
Squiggly
StrikeOut
Redact
Stamp
Caret
Ink
Popup
FileAttachment
Sound
Movie
RichMedia
Widget
Screen
PrinterMark
TrapNet
Watermark
3D
Projection

#### **deleteAnnotation(*annot*)**

Delete the annotation from the page.

**Arguments**

- **annot** – PDFAnnotation.

**getWidgets()**

Return array of all *widgets* on the page.

**Returns**

[].

**update()**

Loop through all annotations of the page and update them. Returns true if re-rendering is needed because at least one annotation was changed (due to either events or *JavaScript* actions or annotation editing).

**applyRedactions(*blackBoxes*, *imageMethod*)**

Applies redactions to the page.

**Arguments**

- **blackBoxes** – Boolean Whether to use black boxes at each redaction or not.
- **imageMethod** – Integer. 0 for no redactions, 1 to redact entire images, 2 for redacting just the covered pixels.

---

**Note:** Redactions are secure as they remove the affected content completely.

---

**process(*processor*)**

Run through the page contents stream and call methods on the supplied *PDF processor*.

**Arguments**

- **processor** – User defined function.

**toPixmap(*transform*, *colorspace*, *alpha*, *renderExtra*, *usage*)**

Render the page into a Pixmap using the given *colorspace* and *alpha* while applying the *transform*. Rendering of annotations/widgets can be disabled. A page can be rendered for e.g. “View” or “Print” usage.

**Arguments**

- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **colorspace** – ColorSpace.
- **alpha** – Boolean.
- **renderExtra** – Boolean.

**Usage**

String.

**getTransform()**

Return the transform from *Fitz* page space (upper left page origin, y descending, 72 dpi) to *PDF* user space (arbitrary page origin, y ascending, UserUnit dpi).

**Returns**

[a,b,c,d,e,f]. The transform *matrix*.

### 3.4.24 PDFAnnotation

*PDF* Annotations belong to a specific *PDFPage* and may be created/changed/removed. Because annotation appearances may change (for several reasons) it is possible to scan through the annotations on a page and query them whether a re-render is necessary. Finally redaction annotations can be applied to a *PDFPage*, destructively removing content from the page.

#### **bound()**

Returns a rectangle containing the location and dimension of the annotation.

##### **Returns**

[ulx,uly,lrx,lry] *Rectangle*.

#### **run(device, transform)**

Calls the device functions to draw the annotation.

##### **Arguments**

- **device** – Device.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.

#### **toPixmap(transform, colorspace, alpha)**

Render the annotation into a *Pixmap*, using the transform and colorspace.

##### **Arguments**

- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **colorspace** – *ColorSpace*.
- **alpha** – Boolean.

#### **toDisplayList()**

Record the contents of the annotation into a *DisplayList*.

##### **Returns**

*DisplayList*.

#### **getObject()**

Get the underlying *PDFObject* for an annotation.

##### **Returns**

*PDFObject*.

#### **process(processor)**

Run through the annotation appearance stream and call methods on the supplied *PDF processor*.

##### **Arguments**

- **processor** – User defined function.

#### **setAppearance(appearance, state, transform, displayList)**

Set the annotation appearance stream for the given appearance. The desired appearance is given as a transform along with a display list.

##### **Arguments**

- **appearance** – String Appearance stream (“N”, “R” or “D”).
- **state** – String “On” or “Off”.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.

- **displayList** – DisplayList.

**setAppearance**(*appearance, state, transform, bbox, resources, contents*)

Set the annotation appearance stream for the given appearance. The desired appearance is given as a transform along with a bounding box, a *PDF* dictionary of resources and a content stream.

#### Arguments

- **appearance** – String Appearance stream (“N”, “R” or “D”).
- **state** – String “On” or “Off”.
- **transform** – [a,b,c,d,e,f]. The transform *matrix*.
- **bbox** – [ulx,uly,lrx,lry] *Rectangle*.
- **resources** – Resources object.
- **contents** – Contents string.

#### Appearance stream values

Value	Description
N	normal appearance
R	roll-over appearance
D	down (pressed) appearance

**update()**

Update the appearance stream to account for changes in the annotation.

**getHot()**

Get the annotation as being hot, *i.e.* that the pointer is hovering over the annotation.

#### Returns

Boolean.

**setHot**(*hot*)

Set the annotation as being hot, *i.e.* that the pointer is hovering over the annotation.

#### Arguments

- **hot** – Boolean.

---

These properties are available for all annotation types.

**getType()**

Return the annotation type.

#### Returns

String *Annotation type*.

**getFlags()**

Get the annotation flags.

#### Returns

Integer which determines the bit-field value.

**setFlags(*flags*)**

Set the annotation flags.

**Arguments**

- **flags** – Integer which determines the bit-field value.

**Annotation flags**

Bit position	Name
1	Invisible
2	Hidden
3	Print
4	NoZoom
5	NoRotate
6	NoView
7	ReadOnly
8	Locked
9	ToggleNoView
10	LockedContents

**getContents()**

Get the annotation contents.

**Returns**

String.

**setContents(*text*)**

Set the annotation contents.

**Arguments**

- **text** – String.

**getBorder()**

Get the annotation border line width in points.

**Returns**

Float.

**setBorder(*width*)**

Set the annotation border line width in points. Use [\*setBorderWidth\(\)\*](#) to avoid removing the border effect.

**Arguments**

- **width** – Float Border width.

**getColor()**

Get the annotation color, represented as an array of up to 4 component values.

**Returns**

The *color value*.

**setColor(*color*)**

Set the annotation color, represented as an array of up to 4 component values.

**Arguments**

- **color** – The *color value*.

**getOpacity()**

Get the annotation opacity.

**Returns**

The *opacity* value.

**setOpacity(*opacity*)**

Set the annotation opacity.

**Arguments**

- **opacity** – The *opacity* value.

**getCreationDate()**

Get the annotation creation date as a *JavaScript* Date object.

**Returns**

Date.

**setCreationDate(*milliseconds*)**

Set the creation date to the number of milliseconds since the epoch.

**Arguments**

- **milliseconds** – Integer Milliseconds value.

**getModificationDate()**

Get the annotation modification date as a *JavaScript* Date object.

**Returns**

Date.

**setModificationDate(*milliseconds*)**

Set the annotation modification date to the number of milliseconds since the epoch.

**Arguments**

- **milliseconds** – Integer Milliseconds value.

**getQuadding()**

Get the annotation quadding (justification).

**Returns**

Quadding value, 0 for left-justified, 1 for centered, 2 for right-justified.

**setQuadding(*value*)**

Set the annotation quadding (justification).

**Arguments**

- **value** – Quadding value, 0 for left-justified, 1 for centered, 2 for right-justified.

**getLanguage()**

Get the annotation language (or the get the inherited document language).

**Returns**

String.

**setLanguage(*language*)**

Set the annotation language.

**Arguments**

- **language** – String.

These properties are only present for some annotation types, so support for them must be checked before use.

#### **getRect()**

Get the annotation bounding box.

##### **Returns**

[ulx,uly,lrx,lry] *Rectangle*.

#### **setRect(rect)**

Set the annotation bounding box.

##### **Arguments**

- **rect** – [ulx,uly,lrx,lry] *Rectangle*.

#### **getDefaultAppearance()**

Get the default text appearance used for free text annotations.

##### **Returns**

{font:String, size:Integer, color:[r,g,b]} Returns an object with the key/value pairs.

#### **setDefaultAppearance(font, size, color)**

Set the default text appearance used for free text annotations.

##### **Arguments**

- **font** – String.
- **size** – Integer.
- **color** – The *color value*.

#### **hasInteriorColor()**

Checks whether the annotation has support for an interior color.

##### **Returns**

Boolean.

#### **getInteriorColor()**

Gets the annotation interior color.

##### **Returns**

The *color value*.

#### **setInteriorColor(color)**

Sets the annotation interior color.

##### **Arguments**

- **color** – The *color value*.

#### **hasAuthor()**

Checks whether the annotation has an author.

##### **Returns**

Boolean.



**getAuthor()**

Gets the annotation author.

**Returns**

String.

**setAuthor(*author*)**

Sets the annotation author.

**Arguments**

- **author** – String.

**hasLineEndingStyles()**

Checks the support for line ending styles.

**Returns**

Boolean.

**getLineEndingStyles()**

Gets the line ending styles object.

**Returns**

{start:String, end:String} Returns an object with the key/value pairs.

**setLineEndingStyles(*start*, *end*)**

Sets the line ending styles object.

**Arguments**

- **start** – String.
- **end** – String.

Line ending names
“None”
“Square”
“Circle”
“Diamond”
“OpenArrow”
“ClosedArrow”
“Butt”
“ROpenArrow”
“RClosedArrow”
“Slash”

**hasIcon()**

Checks the support for annotation icon.

**Returns**

Boolean.

**getIcon()**

Gets the annotation icon.

**Returns**

String.

**setIcon(*name*)**

Sets the annotation icon.

**Arguments**

- **name** – String.

Icon type	Icon name
File attachment	“Graph”
	“PaperClip”
	“PushPin”
	“Tag”
Sound	“Mic”
	“Speaker”
Stamp	“Approved”
	“AsIs”
	“Confidential”
	“Departmental”
	“Draft”
	“Experimental”
	“Expired”
	“Final”
	“ForComment”
	“ForPublicRelease”
	“NotApproved”
	“NotForPublicRelease”
Text	“Sold”
	“TopSecret”
	“Comment”
	“Help”
	“Insert”
	“Key”
	“NewParagraph”
	“Note”
	“Paragraph”

**hasLine()**

Checks the support for annotation line.

**Returns**

Boolean.

**getLine()**

Get line end points, represented by an array of two points, each represented as an [x, y] array.

**Returns**

[[x,y], ...].

**setLine(*endpoints*)**

Set line end points, represented by an array of two points, each represented as an [x, y] array.

**Arguments**

- **endpoints** – [[x,y], ...].

**hasOpen()**

Checks the support for annotation open state.

**Returns**

Boolean.

**isOpen()**

Get annotation open state.

**Returns**

Boolean.

**setIsOpen(*state*)**

Set annotation open state.

**Arguments**

- **state** – Boolean.

---

**Note:** “Open” refers to whether the annotation has an open state or is opened - e.g. A note icon is considered “Open” if the user has clicked on it to view its contents.

---

**hasFilespec()**

Checks support for the annotation file specification.

**Returns**

Boolean.

**getFilespec()**

Gets the file specification object.

**Returns**

Object *File Specification Object*.

**setFilespec(*fileSpecObject*)**

Sets the file specification object.

**Arguments**

- **fileSpecObject** – Object *File Specification object*.
- 

The border drawn around some annotations can be controlled by:

**hasBorder()**

Check support for the annotation border style.

**Returns**

Boolean.

**getBorderStyle()**

Get the annotation border style, either of “Solid” or “Dashed”.

**Returns**

String.

**setBorderStyle**(*style*)

Set the annotation border style, either of “Solid” or “Dashed”.

**Arg**

String.

**getBorderWidth**()

Get the border width in points.

**Returns**

Float.

**setBorderWidth**(*width*)

Set the border width in points. Retain any existing border effects.

**Arguments**

- **width** – Float

**getBorderDashCount**()

Returns the number of items in the border dash pattern.

**Returns**

Integer.

**getBorderDashItem**(*i*)

Returns the length of dash pattern item *i*.

**Arguments**

- **i** – Integer Item index.

**Returns**

Integer.

**setBorderDashPattern**(*dashPattern*)

Set the annotation border dash pattern to the given array of dash item lengths. The supplied array represents the respective line stroke and gap lengths, e.g. [1, 1] sets a small dash and small gap, [2, 1, 4, 1] would set a medium dash, a small gap, a longer dash and then another small gap.

**Arguments**

- **dashpattern** – [Integer, Integer, ...]

**clearBorderDash**()

Clear the entire border dash pattern for an annotation.

**addBorderDashItem**(*length*)

Append an item (of the given length) to the end of the border dash pattern.

**Arguments**

- **length** – Integer.

Annotations that have a border effect allows the effect to be controlled by:

**hasBorderEffect**()

Check support for annotation border effect.

**Returns**

Boolean.

**getBorderEffect()**

Get the annotation border effect, either of “None” or “Cloudy”.

**Returns**

String.

**setBorderEffect(*effect*)**

Set the annotation border effect, either of “None” or “Cloudy”.

**Arg**

String.

**getBorderEffectIntensity()**

Get the annotation border effect intensity.

**Returns**

Integer.

**setBorderEffectIntensity(*intensity*)**

Set the annotation border effect intensity. Recommended values are between 0 and 2 inclusive.

**Arg**

Integer.

Ink annotations consist of a number of strokes, each consisting of a sequence of vertices between which a smooth line will be drawn. These can be controlled by:

**hasInkList()**

Check support for the annotation ink list.

**Returns**

Boolean.

**getInkList()**

Get the annotation ink list, represented as an array of strokes, each an array of points each an array of its X/Y coordinates.

**Returns**

[].

**setInkList(*inkList*)**

Set the annotation ink list, represented as an array of strokes, each an array of points each an array of its X/Y coordinates.

**Arg**

[].

**clearInkList()**

Clear the list of ink strokes for the annotation.

**addInkList(*stroke*)**

To the list of strokes, append a stroke, represented as an array of vertices each an array of its X/Y coordinates.

**Arguments**

- **stroke** – [].

**addInkListStroke()**

Add a new empty stroke to the ink annotation.

### **addInkListStrokeVertex**(*vertex*)

Append a vertex to end of the last stroke in the ink annotation. The vertex is an array of its X/Y coordinates.

#### **Arguments**

- **vertex** – [].

Text markup and redaction annotations consist of a set of quadrilaterals controlled by:

### **hasQuadPoints**()

Check support for the annotation quadpoints.

#### **Returns**

Boolean.

### **getQuadPoints**()

Get the annotation quadpoints, describing the areas affected by text markup annotations and link annotations.

#### **Returns**

[].

### **setQuadPoints**(*quadPoints*)

Set the annotation quadpoints, describing the areas affected by text markup annotations and link annotations.

#### **Arguments**

- **quadPoints** – [].

### **clearQuadPoints**()

Clear the list of quad points for the annotation.

### **addQuadPoint**(*quadpoint*)

Append a single quad point as an array of 8 elements, where each pair are the X/Y coordinates of a corner of the quad.

#### **Arguments**

- **quadpoint** – [].

Polygon and polyline annotations consist of a sequence of vertices with a straight line between them. Those can be controlled by:

### **hasVertices**()

Check support for the annotation vertices.

#### **Returns**

Boolean.

### **getVertices**()

Get the annotation vertices, represented as an array of vertices each an array of its X/Y coordinates.

#### **Returns**

[].

### **setVertices**(*vertices*)

Set the annotation vertices, represented as an array of vertices each an array of its X/Y coordinates.

#### **Arguments**

- **vertices** – [].

### **clearVertices**()

Clear the list of vertices for the annotation.

**addVertex**(*vertex*)

Append a single vertex as an array of its X/Y coordinates.

**Arguments**

- **vertex** – [].

Stamp annotations have the option to set a custom image as its appearance.

**setStampImage**(*image*)

Set a custom image appearance for a stamp annotation.

**Arguments**

- **image** – Image.

### 3.4.25 PDFWidget

**getFieldType**()

Return String indicating type of widget: “button”, “checkbox”, “combobox”, “listbox”, “radiobutton”, “signature” or “text”.

**Returns**

String.

**getFieldFlags**()

Return the field flags. Refer to the *PDF* specification for their meanings.

**Returns**

Integer which determines the bit-field value.

**getRect**()

Get the widget bounding box.

**Returns**

[ulx,uly,lrx,lry] *Rectangle*.

**setRect**(*rect*)

Set the widget bounding box.

**Arguments**

- **rect** – [ulx,uly,lrx,lry] *Rectangle*.

**getMaxLen**()

Get maximum allowed length of the string value.

**Returns**

Integer.

**getValue**()

Get the widget value.

**Returns**

String

**setTextValue**(*value*)

Set the widget string value.

**Arguments**

- **value** – String.

**setChoiceValue**(*value*)

Sets the value against the widget.

**Arguments**

- **value** – String.

**toggle**()

Toggle the state of the widget, returns 1 if the state changed.

**Returns**

Integer.

**getOptions**()

Returns an array of strings which represents the value for each corresponding radio button or checkbox field.

**Returns**

[].

**layoutTextWidget**()

Layout the value of a text widget. Returns a *Text Layout Object*.

**Returns**

Object.

**isReadOnly**()

If the value is read only and the widget cannot be interacted with.

**Returns**

Boolean.

**getLabel**()

Get the field name as a string.

**Returns**

String.

**getEditingState**()

Gets whether the widget is in editing state.

**Returns**

Boolean.

**setEditingState**(*state*)

Set whether the widget is in editing state.

**Arguments**

- **state** – Boolean.

---

**Note:** When in editing state any changes to the widget value will not cause any side-effects such as changing other widgets or running *JavaScript*. This is intended for, e.g. when a text widget is interactively having characters typed into it. Once editing is finished the state should be reverted back, before updating the widget value again.

---

**update**()

Update the appearance stream to account for changes to the widget.



**isSigned()**

Returns *true* if the signature is signed.

**Returns**

Boolean.

**validateSignature()**

Returns number of updates ago when signature became invalid. Returns 0 if signature is still valid, 1 if it became invalid during the last save, etc.

**Returns**

Integer.

**checkCertificate()**

Returns “OK” if signature checked out OK, otherwise a text string containing an error message, e.g. “Self-signed certificate.” or “Signature invalidated by change to document.”, etc.

**Returns**

String.

**getSignatory()**

Returns a text string with the distinguished name from a signed signature, or a text string with an error message.

**Returns**

String.

**previewSignature(*signer, signatureConfig, image, reason, location*)**

Return a *Pixmap* preview of what the signature would look like if signed with the given configuration. Reason and location may be undefined, in which case they are not shown.

**Arguments**

- **signer** – *PDFPKCS7Signer*.
- **signatureConfig** – *Signature Configuration Object*.
- **image** – *Image*.
- **reason** – String.
- **location** – String.

**Returns**

*Pixmap*.

**sign(*signer, signatureConfig, image, reason, location*)**

Sign the signature with the given configuration. Reason and location may be undefined, in which case they are not shown.

**Arguments**

- **signer** – *PDFPKCS7Signer*.
- **signatureConfig** – *Signature Configuration Object*.
- **image** – *Image*.
- **reason** – String.
- **location** – String.

**clearSignature()**

Clear a signed signature, making it unsigned again.

#### **eventEnter()**

Trigger the event when the pointing device enters a widget's active area.

#### **eventExit()**

Trigger the event when the pointing device exits a widget's active area.

#### **eventDown()**

Trigger the event when the pointing device's button is depressed within a widget's active area.

#### **eventUp()**

Trigger the event when the pointing device's button is released within a widget's active area.

#### **eventFocus()**

Trigger the event when the a widget gains input focus.

#### **eventBlur()**

Trigger the event when the a widget loses input focus.

### 3.4.26 PDFPKCS7Signer

#### Creating a Signer

To create a signer object an instance of PDFPKCS7Signer is required.

#### **new(filename, password)**

Read a certificate and private key from a *pfx* file and create a *signer* to hold this information. Used with *PDFWidget.sign()*.

##### Arguments

- **filename** – String.
- **password** – String.

##### Returns

PDFPKCS7Signer.

#### Example

```
var signer = new PDFPKCS7Signer(<file_name>,<password>);
```

### 3.4.27 OutlineIterator

An outline iterator is can be used to walk over all the items in an *Outline* and query their properties. To be able to insert items at the end of a list of sibling items, it can also walk one item past the end of the list.

---

**Note:** In the context of a *PDF* file, the document's *Outline* is also known as *Table of Contents* or *Bookmarks*.

---

#### **item()**

Return an *Outline Iterator Object* or undefined if out of range.

##### Returns

Object.

**next()**

Move the iterator position to “next”.

**Returns**

Int.

**prev()**

Move the iterator position to “previous”.

**Returns**

Int.

**up()**

Move the iterator position “up”.

**Returns**

Int.

**down()**

Move the iterator position “down”.

**Returns**

Int.

---

**Note:** The [next\(\)](#), [prev\(\)](#), [up\(\)](#), [down\(\)](#) methods returns 0 if the new position has a valid item, or 1 if the position contains no valid item, but one may be inserted at the position. If neither of these conditions are met then -1 is returned.

---

**insert(item)**

Insert item before the current item. The position does not change. Returns 0 if the position has a valid item, or 1 if the position has no valid item.

**Arguments**

- **item** – Object which conforms to the *Outline Iterator Object*.

**Returns**

Int.

**delete()**

Delete the current item. This implicitly moves to the next item. Returns 0 if the new position has a valid item, or 1 if the position contains no valid item, but one may be inserted at this position.

**Returns**

Int.

**update(item)**

Updates the current item with the properties of the supplied item.

**Arguments**

- **item** – Object which conforms to the *Outline Iterator Object*.

### 3.4.28 Archive

**new Archive**(*path*)

*Constructor method.*

Create a new archive based either on a *tar* or *zip* file or the contents of a directory.

**Arguments**

- **path** – String.

**Returns**

Archive.

**Example**

```
var archive = new Archive(<path>);
```

**getFormat**()

Returns a string describing the archive format.

**Returns**

String.

**countEntries**()

Returns the number of entries in the archive.

**Returns**

Integer.

**listEntry**(*idx*)

Returns the name of entry number *idx* in the archive.

**Arguments**

- **idx** – Integer.

**Returns**

String.

**hasEntry**(*name*)

Returns *true* if an entry of the given name exists in the archive.

**Arguments**

- **name** – String.

**Returns**

Boolean.

**readEntry**(*name*)

Returns the contents of the entry of the given name.

**Arguments**

- **name** – String.

**Returns**

String.

### 3.4.29 MultiArchive

#### **new MultiArchive()**

*Constructor method.*

Create a new empty multi archive.

#### **Returns**

MultiArchive.

#### **Example**

```
var multiArchive = new MultiArchive();
```

#### **mountArchive(subArchive, path)**

Add an archive to the set of archives handled by a multi archive. If `path` is `null`, the `subArchive` contents appear at the top-level, otherwise they will appear prefixed by the string `path`.

#### **Arguments**

- **subArchive** – Archive.
- **path** – String.

### 3.4.30 TreeArchive

#### **new TreeArchive()**

*Constructor method.*

Create a new empty tree archive.

#### **Returns**

TreeArchive.

#### **Example**

```
var treeArchive = new TreeArchive();
```

#### **add(name, buffer)**

Add a named buffer to a tree archive.

#### **Arguments**

- **name** – String.
- **buffer** – Buffer.

### 3.4.31 Story

#### **new Story(contents, userCSS, em, archive)**

*Constructor method.*

Create a new story with the given `contents`, formatted according to the provided `userCSS` and `em` size, and an archive to lookup images, etc.

#### **Arguments**

- **contents** – String *HTML* source code. If omitted, a basic minimum is generated.

- **userCSS** – String CSS source code. If provided, must contain valid CSS specifications.
- **em** – Float The default text font size.
- **archive** – An Archive from which to load resources for rendering. Currently supported resource types are images and text fonts. If omitted, the Story will not try to look up any such data and may thus produce incomplete output.

**Example**

```
var story = new Story(<contents>, <css>, <em>, <archive>);
```

**document()**

Return an XML for an unplaced story. This allows adding content before placing the Story.

**Returns**

*XML*.

**place(*rect*)**

Place (or continue placing) a Story into the supplied rectangle, returning a *Placement Result Object*. Call *draw()* to draw the placed content before calling *place()* again to continue placing remaining content.

**Arguments**

- **rect** – [*ulx*, *uly*, *lrx*, *lry*] *Rectangle*.

**draw(*device*, *transform*)**

Draw the placed Story to the given device with the given *transform*.

**Arguments**

- **device** – *Device*.
- **transform** – [*a*, *b*, *c*, *d*, *e*, *f*]. The transform *matrix*.

### 3.4.32 XML

This represents an *HTML* or an *XML* node. It is a helper class intended to access the *DOM* (*Document Object Model*) content of a *Story* object.

**body()**

Return an XML for the body element.

**Returns**

*XML*.

**documentElement()**

Return an XML for the top level element.

**Returns**

*XML*.

**createElement(*tag*)**

Create an element with the given tag type, but do not link it into the XML yet.

**Arguments**

- **tag** – String.

**Returns**

*XML*.

**createTextNode**(*text*)

Create a text node with the given text contents, but do not link it into the XML yet.

**Arguments**

- **text** – String.

**Returns**

XML.

**find**(*tag*, *attribute*, *value*)

Find the element matching the tag, *attribute* and value. Set either of those to null to match anything.

**Arguments**

- **tag** – String.
- **attribute** – String.
- **value** – String.

**Returns**

XML.

**findNext**(*tag*, *attribute*, *value*)

Find the next element matching the tag, *attribute* and value. Set either of those to null to match anything.

**Arguments**

- **tag** – String.
- **attribute** – String.
- **value** – String.

**Returns**

XML.

**appendChild**(*dom*, *childDom*)

Insert an element as the last child of a parent, unlinking the child from its current position if required.

**Arguments**

- **dom** – XML.
- **childDom** – XML.

**insertBefore**(*dom*, *elementDom*)

Insert an element before this element, unlinking the new element from its current position if required.

**Arguments**

- **dom** – XML.
- **elementDom** – XML.

**insertAfter**(*dom*, *elementDom*)

Insert an element after this element, unlinking the new element from its current position if required.

**Arguments**

- **dom** – XML.
- **elementDom** – XML.

**remove()**

Remove this element from the XML. The element can be added back elsewhere if required.

**clone()**

Clone this element (and its children). The clone is not yet linked into the XML.

**Returns**

XML.

**firstChild()**

Return the first child of the element as a XML, or null if no child exist.

**Returns**

XML | null.

**parent()**

Return the parent of the element as a XML, or null if no parent exists.

**Returns**

XML | null.

**next()**

Return the next element as a XML, or null if no such element exists.

**Returns**

XML | null.

**previous()**

Return the previous element as a XML, or null if no such element exists.

**Returns**

XML | null.

**addAttribute(*attribute*, *value*)**

Add attribute with the given value, returns the updated element as an XML.

**Arguments**

- **attribute** – String.
- **value** – String.

**Returns**

XML.

**removeAttribute(*attribute*)**

Remove the specified attribute from the element.

**Arguments**

- **attribute** – String.

**attribute(*attribute*)**

Return the element's attribute value as a String, or null if no such attribute exists.

**Arguments**

- **attribute** – String.

**Returns**

String | null.



**getAttributes()**

Returns a dictionary object with properties and their values corresponding to the element's attributes and their values.

**Returns**

{ }.

### 3.4.33 PDF Processor

A *PDF processor* provides callbacks that get called for each *PDF* operator handled by `PDFPage` `process()` & `PDFAnnotation` `process()` methods. The callbacks whose names start with `op_` correspond to each *PDF* operator. Refer to the *PDF* specification for what these do and what the callback arguments are.

**Methods****Main methods**

- `push_resources(resources)`
- `pop_resources()`

**General graphics state callbacks**

- `op_w(lineWidth)`
- `op_j(lineJoin)`
- `op_J(lineCap)`
- `op_M(miterLimit)`
- `op_d(dashPattern, phase)`
- `op_ri(intent)`
- `op_i(flatness)`
- `op_gs(name, extGState)`

**Special graphics state**

- `op_q()`
- `op_Q()`
- `op_cm(a, b, c, d, e, f)`

## Path construction

- `op_m(x, y)`
- `op_l(x, y)`
- `op_c(x1, y1, x2, y2, x3, y3)`
- `op_v(x2, y2, x3, y3)`
- `op_y(x1, y1, x3, y3)`
- `op_h()`
- `op_re(x, y, w, h)`

## Path painting

- `op_S()`
- `op_s()`
- `op_F()`
- `op_f()`
- `op_fstar()`
- `op_B()`
- `op_Bstar()`
- `op_b()`
- `op_bstar()`
- `op_n()`

## Clipping paths

- `op_W()`
- `op_Wstar()`

## Text objects

- `op_BT()`
- `op_ET()`

## Text state

- `op_Tc(charSpace)`
- `op_Tw(wordSpace)`
- `op_Tz(scale)`
- `op_TL(leading)`
- `op_Tf(name, size)`
- `op_Tr(render)`
- `op_Ts(rise)`

## Text positioning

- `op_Td(tx, ty)`
- `op_TD(tx, ty)`
- `op_Tm(a, b, c, d, e, f)`
- `op_Tstar()`

## Text showing

- `op_TJ(textArray) number/string`
- `op_Tj(stringOrByteArray)`
- `op_squote(stringOrByteArray)`
- `op_dquote(wordSpace, charSpace, stringOrByteArray)`

## Type 3 fonts

- `op_d0(wx, wy)`
- `op_d1(wx, wy, llx, lly, urx, ury)`

## Color

- `op_CS(name, colorspace)`
- `op_cs(name, colorspace)`
- `op_SC_color(color)`
- `op_sc_color(color)`
- `op_SC_pattern(name, patternID, color)`  
API not settled, arguments may change in the future.
- `op_sc_pattern(name, patternID, color)`  
API not settled, arguments may change in the future.

- **op\_SC\_shade(name, shade)**  
API not settled, arguments may change in the future.
- **op\_sc\_shade(name, shade)**  
API not settled, arguments may change in the future.
- **op\_G(gray)**
- **op\_g(gray)**
- **op\_RG(r, g, b)**
- **op\_rg(r, g, b)**
- **op\_K(c, m, y, k)**
- **op\_k(c, m, y, k)**

### Shadings, images and XObjects

- **op\_BI(image, colorspace)**
- **op\_sh(name, shade)**  
API not settled, arguments may change in the future.
- **op\_Do\_image(name, image)**
- **op\_Do\_form(xobject, resources)**

### Marked content

- **op\_MP(tag)**
- **op\_DP(tag, raw)**
- **op\_BMC(tag)**
- **op\_BDC(tag, raw)**
- **op EMC()**

### Compatibility

- **op\_BX()**
- **op\_EX()**

## *C API*

This is the *MuPDF* C API guide for developers. In this document we will guide you through the public and stable bits of the *MuPDF* library. This is intended to both provide an introduction to new developers wanting to use low level features of *MuPDF*, and as a reference guide to the public interface.

We will be explaining the basics, enough to get you started on a single threaded application. There are more functions and data structures used internally, and there are also ways to use *MuPDF* from multiple threads, but those are beyond the scope of this document.

The functions and structures documented in this document are what we consider stable *APIs*. They will rarely change, and if they do, any such changes will be noted in the “api-changes” document along with instructions for how to update your code.

## 4.1 *MuPDF* modules

*MuPDF* is separated into several modules.

### 4.1.1 Core API

The core module contains the runtime context, exception handling, and various string manipulation, math, hash table, binary tree, and other useful functions.

#### Core API

Almost all functions in the *MuPDF* library take a `fz_context` structure as their first argument. The context is used for many things; primarily it holds the exception stack for our `setjmp` based exception handling. It also holds various caches and auxiliary contexts for font rendering and color management.

#### The *Fitz* Context

---

**Note:** If you wonder where the prefix “fz” and name *Fitz* come from, *MuPDF* originally started out as a prototype of a new rendering library architecture for *Ghostscript*. It was to be a fusion of *libart* and *Ghostscript*. History turned out differently, and the project mutated into a standalone *PDF* renderer now called *MuPDF*. The “fz” prefix for the graphics library and core modules remains to this day.

---

Here is the code to create a *Fitz* context. The first two arguments are used if you need to use a custom memory allocator, and the third argument is a hint to much memory the various caches should be allowed to grow. The limit is only a soft

limit. We may exceed it, but will start clearing out stale data to try to stay below the limit when possible. Setting it to a lower value will prevent the caches from growing out of hand if you are tight on memory.

```
#include <mupdf/fitz.h>

main()
{
    fzf_context *ctx;

    ctx = fzf_new_context(NULL, NULL, FZF_STORE_UNLIMITED);
    if (!ctx)
        die("Failed to create a new Fitz context!");

    ... do stuff ...

    fzf_drop_context(ctx);
}
```

## Error Handling

*MuPDF* uses a `setjmp` based exception handling system. This is encapsulated by the use of three macros: `fzf_try`, `fzf_always`, and `fzf_catch`. When an error is raised by `fzf_throw`, or re-raised by `fzf_rethrow`, execution will jump to the enclosing `always/catch` block.

All functions you call should be guarded by a `fzf_try` block to catch the errors, or the program will call `exit()` on errors. You don't want that.

The `fzf_always` block is optional. It is typically used to free memory or release resources unconditionally, in both the case when the execution of the `try` block succeeds, and when an error occurs.

```
fzf_try(ctx) {
    // Do stuff that may throw an exception.
}
fzf_always(ctx) {
    // This (optional) block is always executed.
}
fzf_catch(ctx) {
    // This block is only executed when recovering from an exception.
}
```

Since the `fzf_try` macro is based on `setjmp`, the same conditions that apply to local variables in the presence of `setjmp` apply. Any locals written to inside the `try` block may be restored to their pre-try state when an error occurs. We provide a `fzf_var()` macro to guard against this.

In the following example, if we don't guard `buf` with `fzf_var`, then when an error occurs the `buf` local variable might have been reset to its pre-try value (`NULL`) and we would leak the memory.

```
char *buf = NULL;
fzf_var(buf);
fzf_try(ctx) {
    buf = fzf_malloc(ctx, 100);
    // Do stuff with buf that may throw an exception.
}
fzf_always(ctx) {
```

(continues on next page)

(continued from previous page)

```

    fz_free(ctx, buf);
}
fz_catch(ctx) {
    fz_rethrow(ctx);
}

```

Carefully note that you should **never** return from within a `fz_try` or `fz_always` block! Doing so will unbalance the exception stack, and things will go catastrophically wrong. Instead, it is possible to break out of the `fz_try` and `fz_always` block by using a `break` statement if you want to exit the block early without throwing an exception.

Throwing a new exception can be done with `fz_throw`. Passing an exception along after having cleaned up in the `fz_catch` block can be done with `fz_rethrow`. `fz_throw` takes a `printf`-like formatting string.

```

enum {
    FZ_ERROR_MEMORY, // when malloc fails
    FZ_ERROR_SYNTAX, // recoverable syntax errors
    FZ_ERROR_GENERIC, // all other errors
};
void fz_throw(fz_context *ctx, int error_code, const char *fmt, ...);
void fz_rethrow(fz_context *ctx);

```

## Memory Allocation

You should not need to do raw memory allocation using the *Fitz* context, but if you do, here are the functions you need. These work just like the regular *C* functions, but take a *Fitz* context and throw an exception if the allocation fails. They will **not** return `NULL`; either they will succeed or they will throw an exception.

```

void *fz_malloc(fz_context *ctx, size_t size);
void *fz_realloc(fz_context *ctx, void *old, size_t size);
void *fz_calloc(fz_context *ctx, size_t count, size_t size);
void fz_free(fz_context *ctx, void *ptr);

```

There are also some macros that allocate structures and arrays, together with a type cast to catch typing errors.

```

T *fz_malloc_struct(fz_context *ctx, T); // Allocate and zero the memory.
T *fz_malloc_array(fz_context *ctx, size_t count, T); // Allocate uninitialized memory!
T *fz_realloc_array(fz_context *ctx, T *old, size_t count, T);

```

In the rare case that you need an allocation that returns `NULL` on failure, there are variants for that too: `fz_malloc_no_throw`, etc.

## Pool Allocator

The pool allocator is used for allocating many small objects that live and die together. All objects allocated from the pool will be freed when the pool is freed.

```

typedef struct { opaque } fz_pool;

fz_pool *fz_new_pool(fz_context *ctx);
void *fz_pool_alloc(fz_context *ctx, fz_pool *pool, size_t size);
char *fz_pool_strdup(fz_context *ctx, fz_pool *pool, const char *s);
void fz_drop_pool(fz_context *ctx, fz_pool *pool);

```

## Reference Counting

Most objects in *MuPDF* use reference counting to keep track of when they are no longer used and can be freed. We use the verbs “keep” and “drop” to increment and decrement the reference count. For simplicity, we also use the word “drop” for non-reference counted objects (so that in case we change our minds and decide to add reference counting to an object, the code that uses it need not change).

## Hash Table

We have a generic hash table structure with fixed length keys.

The keys and values are not reference counted by the hash table. Callers are responsible for manually taking care of reference counting when inserting and removing values from the table, should that be desired.

```
typedef struct { opaque } fz_hash_table;
```

```
fz_hash_table *fz_new_hash_table(fz_context *ctx, int initial_size, int key_length, int lock, void (*drop_value)(fz_context *ctx, void *value));
```

The lock parameter should be zero, any other value will result in unpredictable behavior. The drop\_value callback function to the constructor is only used to release values when the hash table is destroyed.

```
void fz_drop_hash_table(fz_context *ctx, fz_hash_table *table);
```

Free the hash table and call the drop\_value function on all the values in the table.

```
void *fz_hash_find(fz_context *ctx, fz_hash_table *table, const void *key);
```

Find the value associated with the key. Returns NULL if not found.

```
void *fz_hash_insert(fz_context *ctx, fz_hash_table *table, const void *key, void *value);
```

Insert the value into the hash table. Inserting a duplicate entry will **not** overwrite the old value, it will return the old value instead. Return NULL if the value was inserted for the first time. Does not reference count the value!

```
void fz_hash_remove(fz_context *ctx, fz_hash_table *table, const void *key);
```

Remove the associated value from the hash table. This will not reference count the value!

```
void fz_hash_for_each(fz_context *ctx, fz_hash_table *table, void *state, void (*callback)(fz_context *ctx, void *state, void *key, int key_length, void *value));
```

Iterate and call a function for each key-value pair in the table.

## Binary Tree

The fz\_tree structure is a self-balancing binary tree that maps text strings to values.

```
typedef struct { opaque } fz_tree;
```

```
void *fz_tree_lookup(fz_context *ctx, fz_tree *node, const char *key);
```

Look up an entry in the tree. Returns NULL if not found.

```
fz_tree *fz_tree_insert(fz_context *ctx, fz_tree *root, const char *key, void *value);
```

Insert a new entry into the tree. Do not insert duplicate entries. Returns the new root object.

```
void fz_drop_tree(fz_context *ctx, fz_tree *node, void (*dropfunc)(fz_context *ctx, void *value));
```

Free the tree and all the values in it.

There is no constructor for this structure, since there is no containing root structure. Instead, the insert function returns the new root node. Use NULL for the initial empty tree.



```
fz_tree *tree = NULL;
tree = fz_tree_insert(ctx, tree, "A", my_a_obj);
tree = fz_tree_insert(ctx, tree, "B", my_b_obj);
tree = fz_tree_insert(ctx, tree, "C", my_c_obj);
assert(fz_tree_lookup(ctx, tree, "B") == my_b_obj);
```

## XML Parser

We have a rudimentary *XML* parser that handles well formed *XML*. It does not do any namespace processing, and it does not validate the *XML* syntax.

The parser supports UTF-8, UTF-16, iso-8859-1, iso-8859-7, koi8, windows-1250, windows-1251, and windows-1252 encoded input.

If `preserve_white` is *false*, we will discard all *whitespace-only* text elements. This is useful for parsing non-text documents such as *XPS* and *SVG*. Preserving whitespace is useful for parsing *XHTML*.

```
typedef struct { opaque } fz_xml_doc;
typedef struct { opaque } fz_xml;

fz_xml_doc *fz_parse_xml(fz_context *ctx, fz_buffer *buf, int preserve_white);
void fz_drop_xml(fz_context *ctx, fz_xml_doc *xml);
fz_xml *fz_xml_root(fz_xml_doc *xml);

fz_xml *fz_xml_prev(fz_xml *item);
fz_xml *fz_xml_next(fz_xml *item);
fz_xml *fz_xml_up(fz_xml *item);
fz_xml *fz_xml_down(fz_xml *item);
```

```
int fz_xml_is_tag(fz_xml *item, const char *name);
```

Returns *true* if the element is a tag with the given name.

```
char *fz_xml_tag(fz_xml *item);
```

Returns the tag name if the element is a tag, otherwise *NULL*.

```
char *fz_xml_att(fz_xml *item, const char *att);
```

Returns the value of the tag element's attribute, or *NULL* if not a tag or missing.

```
char *fz_xml_text(fz_xml *item);
```

Returns the UTF-8 text of the text element, or *NULL* if not a text element.

```
fz_xml *fz_xml_find(fz_xml *item, const char *tag);
```

Find the next element with the given tag name. Returns the element itself if it matches, or the first sibling if it doesn't. Returns *NULL* if there is no sibling with that tag name.

```
fz_xml *fz_xml_find_next(fz_xml *item, const char *tag);
```

Find the next sibling element with the given tag name, or *NULL* if none.

```
fz_xml *fz_xml_find_down(fz_xml *item, const char *tag);
```

Find the first child element with the given tag name, or *NULL* if none.

## String Functions

All text strings in *MuPDF* use the UTF-8 encoding. The following functions encode and decode UTF-8 characters, and return the number of bytes used by the UTF-8 character (at most FZ\_UTFMAX).

```
enum { FZ_UTFMAX=4 };
int fz_chartorune(int *rune, const char *str);
int fz_runetochar(char *str, int rune);
```

Since many of the *C* string functions are locale dependent, we also provide our own locale independent versions of these functions. We also have a couple of semi-standard functions like `strsep` and `strlcpy` that we can't rely on the system providing. These should be pretty self explanatory:

```
char *fz_strdup(fz_context *ctx, const char *s);
float fz_strtof(const char *s, char **es);
char *fz_strsep(char **stringp, const char *delim);
size_t fz_strlcpy(char *dst, const char *src, size_t n);
size_t fz_strlcat(char *dst, const char *src, size_t n);
void *fz_memmem(const void *haystack, size_t haystacklen, const void *needle, size_t needlelen);
int fz_strcasecmp(const char *a, const char *b);
```

There are also a couple of functions to process filenames and *URLs*:

```
char *fz_cleanname(char *path);
    Rewrite path in-place to the shortest string that names the same path. Eliminates multiple and trailing slashes,
    and interprets "." and "..".

void fz_dirname(char *dir, const char *path, size_t dir_size);
    Extract the directory component from a path.

char *fz_urldecode(char *url);
    Decode URL escapes in-place.
```

## String Formatting

Our `printf` family handles the common `printf` formatting characters, with a few minor differences. We also support several non-standard formatting characters. The same `printf` syntax is used in the `printf` functions in the *I/O* module as well.

```
size_t fz_vsnprintf(char *buffer, size_t space, const char *fmt, va_list args);
size_t fz_snprintf(char *buffer, size_t space, const char *fmt, ...);
char *fz_asprintf(fz_context *ctx, const char *fmt, ...);
```

**%%, %c, %e, %f, %p, %x, %d, %u, %s**

These behave as usual, but only take padding (+,0,space), width, and precision arguments.

**%g float**

Prints the float in the shortest possible format that won't lose precision, except NaN to 0, +Inf to FLT\_MAX, -Inf to -FLT\_MAX.

**%M fz\_matrix\***

Prints all 6 coefficients in the matrix as %g separated by spaces.

**%R fz\_rect\***

Prints all x0, y0, x1, y1 in the rectangle as %g separated by spaces.

**%P fz\_point\***

Prints *x*, *y* in the point as %g separated by spaces.

**%C int**

Formats character as UTF-8. Useful to print unicode text.

**%q char\***

Formats string using double quotes and C escapes.

**%( char\***

Formats string using parenthesis quotes and *Postscript* escapes.

**%n char\***

Formats string using prefix / and *PDF* name hex-escapes.

## Math Functions

We obviously need to deal with lots of points, rectangles, and transformations in *MuPDF*.

Points are fairly self evident. The `fz_make_point` utility function is for use with *Visual Studio* that doesn't yet support the C99 struct initializer syntax.

```
typedef struct {
    float x, y;
} fz_point;

fz_point fz_make_point(float x, float y);
```

Rectangles are represented by two pairs of coordinates. The *x0*, *y0* pair have the smallest values, and in the normal coordinate space used by *MuPDF* that is the upper left corner. The *x1*, *y1* pair have the largest values, typically the lower right corner.

In order to represent an infinite unbounded area, we use an *x0* that is larger than the *x1*.

```
typedef struct {
    float x0, y0;
    float x1, y1;
} fz_rect;

const fz_rect fz_infinite_rect = { 1, 1, -1, -1 };
const fz_rect fz_empty_rect = { 0, 0, 0, 0 };
const fz_rect fz_unit_rect = { 0, 0, 1, 1 };

fz_rect fz_make_rect(float x0, float y0, float x1, float y1);
```

Our matrix structure is a row-major 3x3 matrix with the last column always [ 0 0 1 ]. This is represented as a struct with six fields, in the same order as in *PDF* and *Postscript*. The identity matrix is a global constant, for easy access.

```
/ a b 0 \
| c d 0 |
\ e f 1 /
```

```
typedef struct {
    float a, b, c, d, e, f;
} fz_matrix;
```

(continues on next page)

(continued from previous page)

```
const fz_matrix fz_identity = { 1, 0, 0, 1, 0, 0 };

fz_matrix fz_make_matrix(float a, float b, float c, float d, float e, float f);
```

Sometimes we need to represent a non-axis aligned rectangular-ish area, such as the area covered by some rotated text. For this we use a quad representation, using a points for each of the upper/lower/left/right corners as seen from the reading direction of the text represented.

```
typedef struct {
    fz_point ul, ur, ll, lr;
} fz_quad;
```

### List of math functions

These are simple mathematical operations that can not throw errors, so do not need a context argument.

**float fz\_abs(float f);**  
Abs for float.

**float fz\_min(float a, float b);**  
Min for float.

**float fz\_max(float a, float b);**  
Max for float.

**float fz\_clamp(float f, float min, float max);**  
Clamp for float.

**int fz\_absi(int i);**  
Abs for integer.

**int fz\_mini(int a, int b);**  
Min for integer.

**int fz\_maxi(int a, int b);**  
Max for integer.

**int fz\_clampi(int i, int min, int max);**  
Clamp for integer.

**int fz\_is\_empty\_rect(fz\_rect r);**  
Returns whether the supplied `fz_rect` is empty.

**int fz\_is\_infinite\_rect(fz\_rect r);**  
Returns whether the supplied `fz_rect` is infinite.

**fz\_matrix fz\_concat(fz\_matrix left, fz\_matrix right);**  
Concat two matrices and returns a new matrix.

**fz\_matrix fz\_scale(float sx, float sy);**  
Scale.

**fz\_matrix fz\_shear(float sx, float sy);**  
Shear.

**fz\_matrix fz\_rotate(float degrees);**  
Rotate.

**fz\_matrix fz\_translate(float tx, float ty);**  
Translate.

```
fz_matrix fz_invert_matrix(fz_matrix matrix);
```

Invert a matrix.

```
fz_point fz_transform_point(fz_point point, fz_matrix m);
```

Transform a matrix via a point.

```
fz_point fz_transform_vector(fz_point vector, fz_matrix m);
```

Transform a matrix via a vector (ignores translation).

```
fz_rect fz_transform_rect(fz_rect rect, fz_matrix m);
```

Transform a `fz_rect` via a `fz_rect`.

```
fz_quad fz_transform_quad(fz_quad q, fz_matrix m);
```

Transform a `fz_quad` via a `fz_quad`.

```
int fz_is_point_inside_rect(fz_point p, fz_rect r);
```

Returns whether the point is inside the supplied `fz_rect`.

```
int fz_is_point_inside_quad(fz_point p, fz_quad q);
```

Returns whether the point is inside the supplied `fz_quad`.

```
fz_matrix fz_transform_page(fz_rect mediabox, float resolution, float rotate);
```

Create a transform matrix to draw a page at a given resolution and rotation. The scaling factors are adjusted so that the page covers a whole number of pixels. Resolution is given in dots per inch. Rotation is expressed in degrees (0, 90, 180, and 270 are valid values).

## 4.1.2 I/O

The I/O module contains structures and functions for buffers of data, reading and writing to streams, compression, and encryption.

### I/O API

#### Buffers

In order to represent a generic chunks of data we use the `fz_buffer` structure.

```
typedef struct {
    unsigned char *data;
    size_t len; // current length
    size_t cap; // total capacity
    ... reserved internal fields ...
} fz_buffer;

fz_buffer *fz_keep_buffer(fz_context *ctx, fz_buffer *buf);
void fz_drop_buffer(fz_context *ctx, fz_buffer *buf);
```

There are many ways to create a buffer. Some create a buffer shared immutable data, others with data you can edit, and others by copying or decoding other data.

```
fz_buffer *fz_new_buffer(fz_context *ctx, size_t capacity);
```

Create a new empty buffer, with the given initial capacity.

```
fz_buffer *fz_new_buffer_from_shared_data(fz_context *ctx, const unsigned char *data,
size_t size);
```

Create a new buffer wrapping the given data pointer. The data is only referenced, and will not be freed when the buffer is destroyed. The data pointer must **not** change or disappear while the buffer lives.

```
fz_buffer *fz_new_buffer_from_copied_data(fz_context *ctx, const unsigned char *data,
size_t size);
```

Create a buffer containing a copy of the data pointed to.

```
fz_buffer *fz_new_buffer_from_base64(fz_context *ctx, const char *data, size_t size);
```

Create a buffer containing the decoded BASE64 data.

Sometimes you want to create buffers piece by piece by appending strings or other data to it, while dynamically growing the underlying storage.

```
void fz_append_data(fz_context *ctx, fz_buffer *buf, const void *data, size_t len);
void fz_append_string(fz_context *ctx, fz_buffer *buf, const char *string);
void fz_append_byte(fz_context *ctx, fz_buffer *buf, int byte);
void fz_append_rune(fz_context *ctx, fz_buffer *buf, int rune);
void fz_append_int16_be(fz_context *ctx, fz_buffer *buf, int x);
void fz_append_int16_le(fz_context *ctx, fz_buffer *buf, int x);
void fz_append_int32_be(fz_context *ctx, fz_buffer *buf, int x);
void fz_append_int32_le(fz_context *ctx, fz_buffer *buf, int x);
void fz_append_printf(fz_context *ctx, fz_buffer *buffer, const char *fmt, ...);
void fz_append_vprintf(fz_context *ctx, fz_buffer *buffer, const char *fmt, va_list_
↪ args);
```

You can also write a bit stream with the following functions. The buffer length always covers all the bits in the buffer, including any unused ones in the last byte, which will always be zero.

```
void fz_append_bits(fz_context *ctx, fz_buffer *buf, int value, int count);
```

Write the lower count bits from value into the buffer.

```
void fz_append_bits_pad(fz_context *ctx, fz_buffer *buf);
```

Write enough zero bits to be byte aligned.

You can use the buffer data as a zero-terminated C string by calling the following function. This will ensure that there is a zero terminator after the last byte and return a pointer to the first byte. This pointer is only borrowed, and should only be used briefly, before the buffer is changed again (which may reallocate or free the data).

```
const char *fz_string_from_buffer(fz_context *ctx, fz_buffer *buf);
```

You can also read and write the contents of a buffer to file.

```
fz_buffer *fz_read_file(fz_context *ctx, const char *filename);
```

Read the contents of a file into a buffer.

```
void fz_save_buffer(fz_context *ctx, fz_buffer *buf, const char *filename);
```

Save the contents of a buffer to a file.

## Input Streams and Filters

An input stream reads data from a source. Some stream types can decompress and decrypt data, and these streams can be chained together in a pipeline.

```
typedef struct { internal } fz_stream;

fz_stream *fz_keep_stream(fz_context *ctx, fz_stream *stm);
void fz_drop_stream(fz_context *ctx, fz_stream *stm);
```

```
fz_stream *fz_open_file(fz_context *ctx, const char *filename);
```

Open a stream reading the contents of a file.

```
fz_stream *fz_open_memory(fz_context *ctx, const unsigned char *data, size_t len);
```

Open a stream reading from the data pointer.

```
fz_stream *fz_open_buffer(fz_context *ctx, fz_buffer *buf);
```

Open a stream reading from a buffer.

The basic stream operations you expect are available.

```
int64_t fz_tell(fz_context *ctx, fz_stream *stm);
void fz_seek(fz_context *ctx, fz_stream *stm, int64_t offset, int whence);
size_t fz_read(fz_context *ctx, fz_stream *stm, unsigned char *data, size_t len);
size_t fz_skip(fz_context *ctx, fz_stream *stm, size_t len);
```

```
fz_buffer *fz_read_all(fz_context *ctx, fz_stream *stm, size_t initial);
```

Read the remaining data into a new buffer.

```
char *fz_read_line(fz_context *ctx, fz_stream *stm, char *buf, size_t n);
```

Behaves like fgets().

```
int fz_read_byte(fz_context *ctx, fz_stream *stm);
int fz_peek_byte(fz_context *ctx, fz_stream *stm);
int fz_is_eof(fz_context *ctx, fz_stream *stm);
```

You can read binary data one integer at a time. The default is big endian, but *LE* versions are also provided.

```
uint16_t fz_read_uint16(fz_context *ctx, fz_stream *stm);
uint32_t fz_read_uint24(fz_context *ctx, fz_stream *stm);
uint32_t fz_read_uint32(fz_context *ctx, fz_stream *stm);
uint64_t fz_read_uint64(fz_context *ctx, fz_stream *stm);

uint16_t fz_read_uint16_le(fz_context *ctx, fz_stream *stm);
uint32_t fz_read_uint24_le(fz_context *ctx, fz_stream *stm);
uint32_t fz_read_uint32_le(fz_context *ctx, fz_stream *stm);
uint64_t fz_read_uint64_le(fz_context *ctx, fz_stream *stm);

int16_t fz_read_int16(fz_context *ctx, fz_stream *stm);
int32_t fz_read_int32(fz_context *ctx, fz_stream *stm);
int64_t fz_read_int64(fz_context *ctx, fz_stream *stm);

int16_t fz_read_int16_le(fz_context *ctx, fz_stream *stm);
int32_t fz_read_int32_le(fz_context *ctx, fz_stream *stm);
int64_t fz_read_int64_le(fz_context *ctx, fz_stream *stm);
```

Reading bit streams is also possible:

```
unsigned int fz_read_bits(fz_context *ctx, fz_stream *stm, int n);
unsigned int fz_read_rbits(fz_context *ctx, fz_stream *stm, int n);
void fz_sync_bits(fz_context *ctx, fz_stream *stm);
int fz_is_eof_bits(fz_context *ctx, fz_stream *stm);
```

Various decoding, decompression, and decryption filters can be chained together.

```
fz_stream *fz_open_null_filter(fz_context *ctx, fz_stream *chain, int len, int64_t
↳ offset);
fz_stream *fz_open_arc4(fz_context *ctx, fz_stream *chain, unsigned char *key, unsigned
↳ keylen);
```

(continues on next page)

(continued from previous page)

```

fz_stream *fz_open_aesd(fz_context *ctx, fz_stream *chain, unsigned char *key, unsigned_
↳keylen);
fz_stream *fz_open_a85d(fz_context *ctx, fz_stream *chain);
fz_stream *fz_open_ahxd(fz_context *ctx, fz_stream *chain);
fz_stream *fz_open_rld(fz_context *ctx, fz_stream *chain);
fz_stream *fz_open_flated(fz_context *ctx, fz_stream *chain, int window_bits);

fz_stream *fz_open_dctd(fz_context *ctx, fz_stream *chain,
    int color_transform,
    int l2factor,
    fz_stream *jpegtables);

fz_stream *fz_open_faxd(fz_context *ctx, fz_stream *chain,
    int k,
    int end_of_line,
    int encoded_byte_align,
    int columns,
    int rows,
    int end_of_block,
    int black_is_1);

fz_stream *fz_open_lzwd(fz_context *ctx, fz_stream *chain,
    int early_change,
    int min_bits,
    int reverse_bits,
    int old_tiff);

fz_stream *fz_open_predict(fz_context *ctx, fz_stream *chain,
    int predictor,
    int columns,
    int colors,
    int bpc);

```

## Output Streams and Filters

Output streams let us write data to a sink, usually a file on disk or a buffer. As with the input streams, output streams can be chained together to compress, encrypt, and encode data.

```
typedef struct { internal } fz_output;
```

Because output may be buffered in the writer, we need a separate close function to ensure that an output stream is properly flushed and any end of data markers are written. This is separate to the drop function, which just frees data. If a writing operation has succeeded, you need to call close on the output stream before dropping it. If you encounter an error while writing data, you can just drop the stream directly, since we couldn't finish writing it and closing it properly would be irrelevant.

```

void fz_close_output(fz_context *ctx, fz_output *out);
void fz_drop_output(fz_context *ctx, fz_output *out);

```

Outputs can be created to write to files or buffers. You can also implement your own data sink by providing a state pointer and callback functions.



```

fz_output *fz_new_output_with_path(fz_context *, const char *filename, int append);
fz_output *fz_new_output_with_buffer(fz_context *ctx, fz_buffer *buf);

fz_output *fz_new_output(fz_context *ctx,
    int buffer_size,
    void *state,
    void (*write)(fz_context *ctx, void *state, const void *data, size_t n),
    void (*close)(fz_context *ctx, void *state),
    void (*drop)(fz_context *ctx, void *state));

```

The usual suspects are available, as well as functions to write integers of various sizes and byte orders.

```
void fz_seek_output(fz_context *ctx, fz_output *out, int64_t off, int whence);
```

Seek to a location in the output. This is not available for all output types.

```
int64_t fz_tell_output(fz_context *ctx, fz_output *out);
```

Tell the current write location of the output stream.

```

void fz_write_data(fz_context *ctx, fz_output *out, const void *data, size_t size);
void fz_write_string(fz_context *ctx, fz_output *out, const char *s);
void fz_write_byte(fz_context *ctx, fz_output *out, unsigned char x);
void fz_write_rune(fz_context *ctx, fz_output *out, int rune);
void fz_write_int16_be(fz_context *ctx, fz_output *out, int x);
void fz_write_int16_le(fz_context *ctx, fz_output *out, int x);
void fz_write_int32_be(fz_context *ctx, fz_output *out, int x);
void fz_write_int32_le(fz_context *ctx, fz_output *out, int x);
void fz_write_printf(fz_context *ctx, fz_output *out, const char *fmt, ...);
void fz_write_vprintf(fz_context *ctx, fz_output *out, const char *fmt, va_list ap);
void fz_write_base64(fz_context *ctx, fz_output *out, const unsigned char *data, size_t,
    ↪size, int newline);

```

Output streams can be chained together to add encryption, compression, and encoding. Note that these do not take ownership of the chained stream, they only write to it. For example, you can write a header, create a compression filter stream, write some data to the filter to compress the data, close the filter and then keep writing more data to the original stream.

```

fz_output *fz_new_arc4_output(fz_context *ctx, fz_output *chain, unsigned char *key,
    ↪size_t keylen);
fz_output *fz_new_ascii85_output(fz_context *ctx, fz_output *chain);
fz_output *fz_new_asciihex_output(fz_context *ctx, fz_output *chain);
fz_output *fz_new_deflate_output(fz_context *ctx, fz_output *chain, int effort, int no_
    ↪header);
fz_output *fz_new_rle_output(fz_context *ctx, fz_output *chain);

```

## File Archives

The archive structure is a read-only collection of files. This is typically a *Zip* file or directory on disk, but other formats are also supported.

```
typedef struct { internal } fz_archive;

void fz_drop_archive(fz_context *ctx, fz_archive *arch);

int fz_is_directory(fz_context *ctx, const char *path);

fz_archive *fz_open_directory(fz_context *ctx, const char *path);
fz_archive *fz_open_archive(fz_context *ctx, const char *filename);
fz_archive *fz_open_archive_with_stream(fz_context *ctx, fz_stream *file);

int fz_count_archive_entries(fz_context *ctx, fz_archive *arch);
const char *fz_list_archive_entry(fz_context *ctx, fz_archive *arch, int idx);

int fz_has_archive_entry(fz_context *ctx, fz_archive *arch, const char *name);
fz_stream *fz_open_archive_entry(fz_context *ctx, fz_archive *arch, const char *name);
fz_buffer *fz_read_archive_entry(fz_context *ctx, fz_archive *arch, const char *name);
```

We can also create new *Zip* archives.

```
typedef struct { internal } fz_zip_writer;

fz_zip_writer *fz_new_zip_writer(fz_context *ctx, const char *filename);
fz_zip_writer *fz_new_zip_writer_with_output(fz_context *ctx, fz_output *out);
void fz_write_zip_entry(fz_context *ctx, fz_zip_writer *zip, const char *name, fz_buffer_
↳ *buf, int compress);
void fz_close_zip_writer(fz_context *ctx, fz_zip_writer *zip);
void fz_drop_zip_writer(fz_context *ctx, fz_zip_writer *zip);
```

### 4.1.3 Graphics

The graphics module contains graphic resource objects like colors, fonts, shadings, and images.

#### Graphics API

This module is a grab bag of various graphics related objects and functions.

##### Colors

Colors throughout *MuPDF* are represented as a float vector associated with a colorspace object. Colorspaces come in many variants, the most common of which are *Gray*, *RGB*, and *CMYK*. We also support *Indexed* (palette), *L\*a\*b\**, *Separation*, *DeviceN*, and *ICC* based colorspaces.

```
enum { FZ_MAX_COLORS = 32 };

enum fz_colorspace_type {
    FZ_COLORSPACE_NONE,
```

(continues on next page)

(continued from previous page)

```

    FZ_COLORSPACE_GRAY,
    FZ_COLORSPACE_RGB,
    FZ_COLORSPACE_BGR,
    FZ_COLORSPACE_CMYK,
    FZ_COLORSPACE_LAB,
    FZ_COLORSPACE_INDEXED,
    FZ_COLORSPACE_SEPARATION,
};

struct fz_colorspace_s {
    enum fz_colorspace_type type;
    int n;
    char *name;
    private internal fields
};

fz_colorspace *fz_keep_colorspace(fz_context *ctx, fz_colorspace *colorspace);
void fz_drop_colorspace(fz_context *ctx, fz_colorspace *colorspace);

```

Most colorspaces have color components between 0 and 1. The number of components a color uses is given by the ‘n’ field of the colorspace. This is at most FZ\_MAX\_COLORS.

$L^*a^*b^*$  colors have a range of 0..100 for the  $L^*$  and -128..127 for  $a^*$  and  $b^*$ .

CMYK, Separation, and DeviceN colorspaces are subtractive colorspaces. Separation and DeviceN colorspaces also have a tint transform function and a base colorspace used to represent the colors when rendering to a device that does not have these specific colorants.

## Color management engine

MuPDF is usually built with a color management engine, but this can be turned off at runtime if you need faster performance at the cost of much more inaccurate color conversions.

```

int fz_enable_icc(fz_context *ctx);
int fz_disable_icc(fz_context *ctx);

```

The various color conversion functions also take certain parameters that the color management engine uses, such as rendering intent and overprint control.

```

enum {
    FZ_RI_PERCEPTUAL,
    FZ_RI_RELATIVE_COLORIMETRIC,
    FZ_RI_SATURATION,
    FZ_RI_ABSOLUTE_COLORIMETRIC,
};

typedef struct {
    int ri; // Rendering intent
    int bp; // Black point compensation
    int op;
    int opm;
} fz_color_params;

```

(continues on next page)

(continued from previous page)

```
const fz_color_params fz_default_color_params = { FZ_RI_RELATIVE_COLORIMETRIC, 1, 0, 0 };
```

## Device colorspaces

When you need to define a color, and don't care too much about the calibration, you can use the *Device* colorspaces. When *MuPDF* is built with a color management engine and *ICC* is enabled the *Gray* and *RGB* colorspaces use an *sRGB ICC* profile.

```
fz_colorspace *fz_device_gray(fz_context *ctx);
fz_colorspace *fz_device_rgb(fz_context *ctx);
fz_colorspace *fz_device_bgr(fz_context *ctx);
fz_colorspace *fz_device_cmyk(fz_context *ctx);
fz_colorspace *fz_device_lab(fz_context *ctx);
```

*BGR* is present to allow you to render to pixmaps that have the *RGB* components in a different order, so that the data can be passed directly to the operating system for drawing without needing yet another conversion step.

## Indexed colorspaces

*Indexed* colors have a range of  $0..N$  where  $N$  is one less than the number of colors in the palette. An indexed colorspace also has a base colorspace, which is used to define the palette of colors used.

```
fz_colorspace *fz_new_indexed_colorspace(fz_context *ctx,
    fz_colorspace *base,
    int high,
    unsigned char *lookup);
```

High is the maximum value in the palette; i.e. one less than the number of colors.

The lookup argument is a packed array of color values in the base colorspace, represented as bytes mapped to the range of  $0..255$ .

## ICC colorspaces

You can create *ICC* colorspaces from a buffer containing the *ICC* profile.

```
fz_colorspace *fz_new_icc_colorspace(fz_context *ctx,
    enum fz_colorspace_type type,
    int flags,
    const char *name,
    fz_buffer *buf);
```

The type argument can be *NONE* if you want to automatically infer the colorspace type from the profile data. If the type is anything else, then an error will be thrown if the profile does not match the type.

## Color converters

There are several ways to convert colors. The easiest is to call a function, but if you are converting many colors at once, it will be faster to use a color converter object.

```
void fz_convert_color(fz_context *ctx,
fz_colorspace *src_colorspace,
const float *src_color,
fz_colorspace *dst_colorspace,
float *dst_color,
fz_colorspace *proof_colorspace,
const fz_color_params params);
typedef struct {
void (*convert)(fz_context *ctx, fz_color_converter *cc, const float *src, float *dst);
private internal fields
} fz_color_converter;

void fz_find_color_converter(fz_context *ctx, fz_color_converter *cc,
fz_colorspace *src_colorspace,
fz_colorspace *dst_colorspace,
fz_colorspace *proof_colorspace,
fz_color_params params);
void fz_drop_color_converter(fz_context *ctx, fz_color_converter *cc);
```

Here is some sample code to do a one-off *CMYK* to *RGB* color conversion.

```
float cmyk[4] = { 1, 0, 0, 0 };
float rgb[3];
fz_convert_color(ctx, fz_device_cmyk(ctx), cmyk, fz_device_rgb(ctx), rgb, NULL, fz_
↳default_color_params(ctx));
```

Here is some sample code to do repeated *CMYK* to *RGB* color conversions on many colors using a color converter object.

```
float cmyk[100][4] = { {1,0,0,0}, ...
float rgb[100][3];
int i;
fz_color_converter cc;
fz_find_color_converter(ctx, &cc, fz_device_cmyk(ctx), fz_device_rgb(ctx), NULL, fz_
↳default_color_params(ctx));
for (i = 0; i < 100; ++i)
    cc.convert(ctx, &cc, cmyk[i], rgb[i]);
fz_drop_color_converter(ctx, &cc);
```

## Pixmaps

A pixmap is an 8-bit per component raster image. Each pixel is packed with process colorants, spot colors, and alpha channel in that order. If an alpha channel is present, the process colorants are pre-multiplied with the alpha value.

```
typedef struct {
    int w, h; // Width and height
    int x, y; // X and Y offset
    int n; // Number of components in total (colors + spots + alpha)
    int s; // Number of components that are spot colors
    int alpha; // True if alpha channel is present
    int stride; // Number of bytes per row
    int xres, yres; // Resolution in dots per inch.
    fz_colorspace *colorspace; // Colorspace of samples, or NULL if alpha only pixmap.
    unsigned char *samples;
    private internal fields
} fz_pixmap;

fz_pixmap *fz_keep_pixmap(fz_context *ctx, fz_pixmap *pix);
void fz_drop_pixmap(fz_context *ctx, fz_pixmap *pix);
```

There are too many pixmap constructors. Here is the only one you should need.

```
fz_pixmap *fz_new_pixmap(fz_context *ctx, fz_colorspace *cs, int w, int h, fz_
↳separations *seps, int alpha);
```

A newly created pixmap has uninitialized data. The samples must either be cleared or overwritten with existing data before the pixmap can be safely used.

```
void fz_clear_pixmap(fz_context *ctx, fz_pixmap *pix);
    Clear the pixmap to black.
```

```
void fz_clear_pixmap_with_value(fz_context *ctx, fz_pixmap *pix, int value);
    Clear the pixmap to a grayscale value 0..255, where 0 is black and 255 is white. The value is automatically
    inverted for subtractive colorspaces.
```

```
void fz_fill_pixmap_with_color(fz_context *ctx, fz_pixmap *pix, fz_colorspace
*colorspace, float *color, fz_color_params color_params);
    Fill the pixmap with a solid color.
```

```
void fz_unpack_tile(fz_context *ctx, fz_pixmap *dst, unsigned char *src, int n, int
depth, size_t stride, int scale);
    Unpack pixel values from source data to fill in the pixmap samples. n is the number of samples per pixel, depth
    is the bit depth (1, 2, 4, 8, 16, 24, or 32), stride is the number of bytes per row. If scale is non-zero, it is
    the scaling factor to apply to the input samples to map them to the 8-bpc pixmap range. Pass 1 to the scale for
    indexed images, and 0 for everything else. If there are more components in the source data than the destination,
    they will be dropped. If there are fewer components in the source data, the pixmap will be padded with 255.
```

Some functions can create a pixmap and initialize its samples in one go:

```
fz_pixmap *fz_new_pixmap_from_8bpp_data(fz_context *ctx, int x, int y, int w, int h,
↳unsigned char *data, int stride);
fz_pixmap *fz_new_pixmap_from_1bpp_data(fz_context *ctx, int x, int y, int w, int h,
↳unsigned char *data, int stride);
```

Pixmaps can be tinted, inverted, scaled, gamma corrected, and converted to other colorspaces.

```
void fz_invert_pixmap(fz_context *ctx, fz_pixmap *pix);
```

Invert the pixmap samples.

```
void fz_tint_pixmap(fz_context *ctx, fz_pixmap *pix, int black, int white);
```

Map black to black and white to white. The black and white colors are represented as a packed *RGB* integer. `0xFFFFFFFF` is white, `0xFF0000` is red, and `0x000000` is black.

```
void fz_gamma_pixmap(fz_context *ctx, fz_pixmap *pix, float gamma);
```

Apply a gamma correction curve on the samples. A typical use is to adjust the gamma curve on an inverted image by applying a correction factor of  $1/1.4$ .

```
fz_pixmap *fz_convert_pixmap(fz_context *ctx, fz_pixmap *source_pixmap, fz_colorspace
*destination_colorspace, fz_colorspace *proof_colorspace, fz_default_colorspaces
*default_cs, fz_color_params color_params, int keep_alpha);
```

Convert the source pixmap into the destination colorspace. Pass `NULL` for the `default_cs` parameter.

```
fz_pixmap *fz_scale_pixmap(fz_context *ctx, fz_pixmap *src, float x, float y, float w,
float h, const fz_irect *clip);
```

Scale the pixmap up or down in size to fit the rectangle. Will return `NULL` if the scaling factors are out of range. This applies fancy filtering and will anti-alias the edges for subpixel positioning if using non-integer coordinates. If the clip rectangle is set, the returned pixmap may be subset to fit the clip rectangle. Pass `NULL` to the clip if you want the whole pixmap scaled.

#### 4.1.4 Device

The device interface is how we provide access to the contents of a document. A device is a callback structure, that gets called for each piece of text, line art, and image on a page. There are several device implementations. The most important one renders the contents to a raster image, and another one gathers all the text into a structure that can be used to select and copy and search the text content on a page.

#### 4.1.5 Document

The document module handles reading and writing documents in various formats, and ties together all the preceding modules to provide rendering, format conversion, and search functionality for the document types we support.

#### 4.1.6 PDF

The PDF module provides access to the low level PDF structure, letting you query, modify, and create PDF objects and streams. It allows you to create new documents, modify existing documents, or examine features, extract data, or do almost anything you could want at the PDF object and stream level that we don't provide with the higher level APIs.





## LANGUAGE BINDINGS

Auto-generated *C++*, *Python* and *C#* versions of the *MuPDF C API* are available.

These *APIs* are currently a beta release and liable to change.

### 5.1 The C++ MuPDF API

#### 5.1.1 Basics

- Auto-generated from the MuPDF C API's header files.
- Everything is in C++ namespace `mupdf`.
- All functions and methods do not take `fz_context*` arguments. (Automatically-generated per-thread contexts are used internally.)
- All MuPDF `setjmp()/longjmp()`-based exceptions are converted into C++ exceptions.

#### 5.1.2 Low-level C++ API

The MuPDF C API is provided as low-level C++ functions with `ll_` prefixes.

- No `fz_context*` arguments.
- MuPDF exceptions are converted into C++ exceptions.

#### 5.1.3 Class-aware C++ API

C++ wrapper classes wrap most `fz_*` and `pdf_*` C structs.

- Class names are camel-case versions of the wrapped struct's name, for example `fz_document`'s wrapper class is `mupdf::FzDocument`.
- Classes automatically handle reference counting of the underlying C structs, so there is no need for manual calls to `fz_keep_*`() and `fz_drop_*`(), and class instances can be treated as values and copied arbitrarily.

Class-aware functions and methods take and return wrapper class instances instead of MuPDF C structs.

- No `fz_context*` arguments.
- MuPDF exceptions are converted into C++ exceptions.
- Class-aware functions have the same names as the underlying C API function.
- Args that are pointers to a MuPDF struct will be changed to take a reference to the corresponding wrapper class.

- Where a MuPDF function returns a pointer to a struct, the class-aware C++ wrapper will return a wrapper class instance by value.
- Class-aware functions that have a C++ wrapper class as their first parameter are also provided as a member function of the wrapper class, with the same name as the class-aware function.
- Wrapper classes are defined in `mupdf/platform/c++/include/mupdf/classes.h`.
- Class-aware functions are declared in `mupdf/platform/c++/include/mupdf/classes2.h`.

Usually it is more convenient to use the class-aware C++ API rather than the low-level C++ API.

### 5.1.4 Example wrappers

The MuPDF C API function `fz_new_buffer_from_page()` is available as these C++ functions/methods:

```
// MuPDF C function.
fz_buffer *fz_new_buffer_from_page(fz_context *ctx, fz_page *page, const fz_stext_
↳options *options);

// MuPDF C++ wrappers.
namespace mupdf
{
    // Low-level wrapper:
    ::fz_buffer *ll_fz_new_buffer_from_page(::fz_page *page, const ::fz_stext_options_
↳options);

    // Class-aware wrapper:
    FzBuffer fz_new_buffer_from_page(const FzPage& page, FzStextOptions& options);

    // Method in wrapper class FzPage:
    struct FzPage
    {
        ...
        FzBuffer fz_new_buffer_from_page(FzStextOptions& options);
        ...
    };
}
```

### 5.1.5 Extensions beyond the basic C API

- Some generated classes have extra `begin()` and `end()` methods to allow standard C++ iteration:

```
#include "mupdf/classes.h"
#include "mupdf/functions.h"

#include <iostream>

void show_stext(mupdf::FzStextPage& page)
{
    for (mupdf::FzStextPage::iterator it_page: page)
    {
```

(continues on next page)

(continued from previous page)

```

mupdf::FzStextBlock block = *it_page;
for (mupdf::FzStextBlock::iterator it_block: block)
{
    mupdf::FzStextLine line = *it_block;
    for (mupdf::FzStextLine::iterator it_line: line)
    {
        mupdf::FzStextChar stextchar = *it_line;
        fz_stext_char* c = stextchar.m_internal;
        using namespace mupdf;
        std::cout << "FzStextChar("
            << "c=" << c->c
            << " color=" << c->color
            << " origin=" << c->origin
            << " quad=" << c->quad
            << " size=" << c->size
            << " font_name=" << c->font->name
            << "\n";
    }
}
}
}

```

- There are various custom class methods and constructors.
- There are extra functions for generating a text representation of ‘POD’ structs and their C++ wrapper classes.

For example for `fz_rect` we provide these functions:

```

std::ostream& operator<< (std::ostream& out, const fz_rect& rhs);
std::ostream& operator<< (std::ostream& out, const FzRect& rhs);
std::string to_string_fz_rect(const fz_rect& s);
std::string to_string(const fz_rect& s);
std::string Rect::to_string() const;

```

These each generate text such as: (x0=90.51 y0=160.65 x1=501.39 y1=1215.6)

## 5.1.6 Environmental variables

### All builds

- **MUPDF\_mt\_ctx**

Controls support for multi-threading on startup.

- If set with value 0, a single `fz_context*` is used for all threads; this might give a small performance increase in single-threaded programmes, but will be unsafe in multi-threaded programmes.
- Otherwise each thread has its own `fz_context*`.

One can instead call `mupdf::reinit_singlethreaded()` on startup to force single-threaded mode. This should be done before any other use of MuPDF.

## Debug builds only

Debug builds contain diagnostics/checking code that is activated via these environmental variables:

- **MUPDF\_check\_refs**  
If 1, generated code checks MuPDF struct reference counts at runtime.
- **MUPDF\_trace**  
If 1, generated code outputs a diagnostic each time it calls a MuPDF function (apart from keep/drop functions).  
If 2, we also show arg POD and pointer values.
- **MUPDF\_trace\_director**  
If 1, generated code outputs a diagnostic when doing special handling of MuPDF structs containing function pointers.
- **MUPDF\_trace\_exceptions**  
If 1, generated code outputs diagnostics when it converts MuPDF `setjmp()/longjmp()` exceptions into C++ exceptions.
- **MUPDF\_trace\_keepdrop**  
If 1, generated code outputs diagnostics for calls to `*_keep_*`() and `*_drop_*`().

### 5.1.7 Limitations

- Global instances of C++ wrapper classes are not supported.

This is because:

- C++ wrapper class destructors generally call MuPDF functions (for example `fz_drop_*`()).
- The C++ bindings use internal thread-local objects to allow per-thread `fz_context*`'s to be efficiently obtained for use with underlying MuPDF functions.
- C++ globals are destructed *after* thread-local objects are destructed.

So if a global instance of a C++ wrapper class is created, its destructor will attempt to get a `fz_context*` using internal thread-local objects which will have already been destroyed.

We attempt to display a diagnostic when this happens, but this cannot be relied on as behaviour is formally undefined.

## 5.2 The Python and C# MuPDF APIs

- A Python module called `mupdf`.
- A C# namespace called `mupdf`.
  - C# bindings are experimental as of 2021-10-14.
- Auto-generated from the C++ MuPDF API using SWIG, so inherits the abstractions of the C++ API:
  - No `fz_context*` arguments.
  - Automatic reference counting, so no need to call `fz_keep_*`() or `fz_drop_*`(), and we have value-semantics for class instances.
  - Native Python and C# exceptions.

- Output parameters are returned as tuples.
- Allows implementation of mutool in Python - see `mupdf:scripts/mutool.py` and `mupdf:scripts/mutool_draw.py`.
- Provides text representation of simple ‘POD’ structs:

```
rect = mupdf.FzRect(...)
print(rect) # Will output text such as: (x0=90.51 y0=160.65 x1=501.39 y1=215.6)
```

- This works for classes where the C++ API defines a `to_string()` method as described above.
  - \* Python classes will have a `__str__()` method, and an identical `__repr__()` method.
  - \* C# classes will have a `ToString()` method.
- Uses SWIG Director classes to allow C function pointers in MuPDF structs to call Python code.
  - This has not been tested on C#.

## 5.3 Installing the Python mupdf module using pip

The Python `mupdf` module is available on the [Python Package Index \(PyPI\)](https://pypi.org/) website.

- Install with: `pip install mupdf`
- Pre-built Wheels (binary Python packages) are provided for Windows and Linux.
- For more information on the latest release, see changelog below and: <https://pypi.org/project/mupdf/>

## 5.4 Doxygen/Pydoc API documentation

Auto-generated documentation for the C, C++ and Python APIs is available at: <https://ghostscript.com/~julian/mupdf-bindings/>

- All content is generated from the comments in MuPDF header files.
- This documentation is generated from an internal development tree, so may contain features that are not yet publicly available.
- It is updated only intermittently.

## 5.5 Example client code

### 5.5.1 Using the Python API

Minimal Python code that uses the `mupdf` module:

```
import mupdf
document = mupdf.FzDocument('foo.pdf')
```

A simple example Python test script (run by `scripts/mupdfwrap.py -t`) is:

- `scripts/mupdfwrap_test.py`

More detailed usage of the Python API can be found in:

- `scripts/mutool.py`

- `scripts/mutool_draw.py`

Example Python code that shows all available information about a document's Text blocks, lines and characters.

```
#!/usr/bin/env python3

import mupdf

def show_text(document):
    """
    Shows all available information about Stext blocks, lines and characters.
    """
    for p in range(document.fz_count_pages()):
        page = document.fz_load_page(p)
        stextpage = mupdf.FzStextPage(page, mupdf.FzStextOptions())
        for block in stextpage:
            block_ = block.m_internal
            log(f'block: type={block_.type} bbox={block_.bbox}')
            for line in block:
                line_ = line.m_internal
                log(f'    line: wmode={line_.wmode}'
                    + f' dir={line_.dir}'
                    + f' bbox={line_.bbox}'
                    )
                for char in line:
                    char_ = char.m_internal
                    log(f'        char: {chr(char_.c)!r} c={char_.c:4} color=
→ {char_.color}'
                        + f' origin={char_.origin}'
                        + f' quad={char_.quad}'
                        + f' size={char_.size:6.2f}'
                        + f' font=(
                            + f' is_mono={char_.font.flags.is_mono}'
                            + f' is_bold={char_.font.flags.is_bold}'
                            + f' is_italic={char_.font.flags.is_italic}'
                            + f' ft_substitute={char_.font.flags.ft_
→ substitute}'
                                + f' ft_stretch={char_.font.flags.ft_stretch}'
                                + f' fake_bold={char_.font.flags.fake_bold}'
                                + f' fake_italic={char_.font.flags.fake_italic}'
                                + f' has_opentype={char_.font.flags.has_
→ opentype}'
                                    + f' invalid_bbox={char_.font.flags.invalid_
→ bbox}'
                                        + f' name={char_.font.name}'
                                        + f')'
                                )
                    )

document = mupdf.FzDocument('foo.pdf')
show_text(document)
```

## 5.5.2 Basic PDF viewers written in Python and C#

- `scripts/mupdfwrap_gui.py`
- `scripts/mupdfwrap_gui.cs`
- Build and run with:
  - `./scripts/mupdfwrap.py -b all --test-python-gui`
  - `./scripts/mupdfwrap.py -b --csharp all --test-csharp-gui`

## 5.6 Changelog

[Note that this is only for changes to the generation of the C++/Python/C# APIs; changes to the main MuPDF API are not detailed here.]

- **2023-02-14:**
  - Simplified builds by requiring a standalone libclang (typically pypi.org's libclang in a Python venv) and fixed various issues with using latest libclang.
  - Added test for exceptions from Python SWIG Director callbacks.
- **2023-02-03:**
  - Provide a default constructor for all wrapper classes.
  - Added Python `__repr__()` methods for POD classes, identical to the existing `__str__()` methods.
  - Fixed handling of exceptions in Python SWIG Director callbacks.
  - Fixed wrapping of PDF filters.
- **2023-01-20:**
  - Don't disable SWIG Directors on Windows.
  - Show warnings if env settings (e.g. `MUPDF_trace`) will be ignored because we are a release build.
  - Added Python support for MuPDF Stories.
- **2023-01-12:** New release of Python package **mupdf-1.21.1.20230112.1504** (from **mupdf-1.21.x** git 04c75ec9db31), with pre-built Wheels for Windows and Linux. See: <https://pypi.org/project/mupdf>
  - Reduced size of Python sdist by excluding some test directories.
  - Python installation with `pip` will now automatically install `libclang` and `swig`.
  - Added Windows-specific documentation.
  - Fixes for Windows builds.
- **2022-11-23:**
  - Avoid need to specify `LD_LIBRARY_PATH` on Unix by using `rpath`.
  - Allow misc prefixes in build directory.
  - Added accessors to `fz_text_span` wrapper class. This simplifies use from Python, e.g. returning class wrappers for `.font` and `.trm` members, and giving access to the `.items[]` array.
  - Improved control over single-threaded behaviour.
  - Fixed python wrappers for `fz_set_warning_callback()` and `fz_set_error_callback()`.

- Fixed implementation of `ll_pdf_set_annot_color()`.
- **2022-10-21:**
  - Document that global instances of wrapper classes are not supported.
  - Python: provide class-aware out-param wrappers.
  - Generate `operator==` and `operator!=` for POD structs and wrapper classes.
  - Moved `operator<<` into top-level namespace.
  - Document that we require the `clang` package on Linux when building.
  - Disable unhelpful SWIG warnings when building.
  - Support for calling `fz_document_handler fnptrs` in C++ API.
  - Work around Memento build problem on Linux.
  - Fixed some leaks by improving detection of functions returning kept/borrowed references.
  - Fixed handling of kept/borrowed references in Python/C# functions with out-params.
- **2022-08-29:** Simplified naming of C++/Python/C# classes and functions.
  - Don't remove leading `fz_` from function/method names.
  - For low-level wrappers, add `ll_` prefix to the original name; don't remove initial `fz_`; don't add `p` prefix for `pdf_*()` wrappers.
  - For class-aware wrapper functions, use original C name; don't use `m` prefix.
  - Include initial `Fz` prefix for wrapper classes of `fz_*` structs.

So new naming scheme is:

- Low-level wrappers: prepend `ll_` to the full MuPDF C function name.
- Wrapper class names: convert the full struct name to camel-case.
- Wrapper class methods: use the full wrapped MuPDF C function name.
- Class-aware wrappers: use the full wrapped MuPDF C function name.
- **2022-5-11:** Documented the experimental C# API.
- **2022-3-26:** New release of Python package **mupdf-1.19.0.20220326.1214** (from **mupdf-1.19.0** git 466e06fc7e01), with pre-built Wheels for Windows and Linux. See: <https://pypi.org/project/mupdf/>
  - Fixed SWIG Directors wrapping classes on Windows.
- **2022-3-23:** New release of Python package **mupdf-1.19.0.20220323.1255** (from **mupdf-1.19.0** git 58e2b82bf7d1e7), with pre-built Wheels for Windows and Linux. See: <https://pypi.org/project/mupdf/>

#### Details

- Use SWIG Director classes to support MuPDF structs that contain fn pointers. This allows MuPDF to call Python callback code. [.line-through]#Only available on Unix at the moment.#
  - \* This allows us to provide Python wrappers for `fz_set_warning_callback()` and `fz_set_error_callback()`.
- Added alternative wrappers for MuPDF functions in the form of free-standing functions that operate on our wrapper classes. Useful when porting existing code to Python, and generally as a non-class-based API that still gives automatic handling of reference counting. New functions have same name as underlying MuPDF function with a `m` prefix; they do not take a `fz_context` arg and take/return references to wrapper classes instead of pointers to MuPDF structs.



- \* Class methods now call these new free-standing wrappers.
  - Various improvements to enums and non-copyable class wrappers.
  - Use `/** ... */` comments in generated code so visible to Doxygen.
  - Improvements to and fixes to reference counting.
    - \* Use MuPDF naming conventions for detection of MuPDF functions that return borrowed references.
    - \* Improved detection of whether a MuPDF struct uses reference counting.
    - \* Fixed some reference counting issues when handling out-params.
  - Added optional runtime ref count checking.
  - For fns that return raw unsigned char array, provide C++ wrappers that return a `std::vector`. This works much better with SWIG.
  - Allow construction of `Document` from `PdfDocument`.
  - Allow writes to `PdfWriteOptions::opwd_utf8` and `PdfWriteOptions::upwd_utf8`.
  - Added `Page::doc()` to return wrapper for `.doc` member.
  - Added `PdfPage::super()` to return `Page` wrapper for `.super`.
  - Added `PdfDocument::doc()` to return wrapper for `.doc` member.
  - Added `PdfObj::obj()` to return wrapper for `.obj` member.
  - Made Python wrappers for `fz_fill_text()` take Python tuple/list for `float* color` arg.
  - Improved wrapping of `pdf_lexbuf`.
  - Added `Page` downcast constructor from `PdfPage`.
  - Expose `pdf_widget_type` enum.
  - Improved python bindings for `*dict_getl()` and `*dict_putl()`. We now also provide `mupdf_dict_getl()` etc handling variable number of args.
  - Improvements to wrapping of `pdf_filter_options`, `pdf_redact_options`, `fz_pixmap`, `pdf_set_annot_color`, `pdf_obj`.
  - Allow direct use of `PDF_ENUM_NAME_*` enums as `PdfObj`'s in Python.
  - Added wrappers for `pdf_annot_type()` and `pdf_string_from_annot_type()`.
  - `Buffer.buffer_storage()` raises an exception with useful error info (it is not possible to use it from SWIG bindings).
  - Added various fns to give Python access to some raw pointer values, e.g. for passing to `mupdf.new_buffer_from_copied_data()`.
  - Avoid excluding class method wrappers for `pdf_*` fns in python.
- **2022-02-05:** Uploaded Doxygen/Pydoc documentation for the C, C++ and Python APIs, from latest development tree.
  - **2021-09-29:** Released Python bindings for **mupdf-1.19.0** (git 61b63d734a7) to pypi.org (**mupdf 1.19.0.20210929.1226**) with pre-built Wheels for Windows and Linux.
  - **2021-08-05:** Released Python package **mupdf-1.18.0.20210805.1716** on pypi.org with pre-built Wheels for Windows and Linux.
    - Improved constructors of `fz_document_writer` wrapper class `DocumentWriter`.

- Fixed operator<< for POD C structs - moved from `mupdf` namespace to top-level.
- Added `scripts/mupdfwrap_gui.py` - a simple demo Python PDF viewer.
- Cope with `fz_paint_shade()`'s new `fz_shade_color_cache` `**cache` arg.
- **2021-05-21**: First release of Python package, **mupdf-1.18.0.20210521.1738**, on `pypi.org` with pre-built Wheels for Windows and Linux.

### Details

\* Changes that apply to both C++ and Python bindings:

- Improved access to metadata - added `Document::lookup_metadata()` overload that returns a `std::string`. Also provided `extern const std::vector<std::string> metadata_keys;` containing a list of the supported keys.
- Iterating over `Outline`'s now returns `OutlineIterator` objects so that depth information is also available.
- Fixed a reference-counting bug in iterators.
- `Page::search_page()` now returns a `std::vector`.
- `PdfDocument` now has a default constructor which uses `pdf_create_document()`.
- Include wrappers for functions that return `fz_outline*`, e.g. `Outline Document::load_outline();`.
- Removed potentially slow call of `getenv("MUPDF_trace")` in every C++ wrapper function.
- Removed special-case naming of wrappers for `fz_run_page()` - they are now called `mupdf::run_page()` and `mupdf::Page::run_page()`, not `mupdf::run()` etc.
- Added text representation of POD structs.
- Added support for 32 and 64-bit Windows.
- Many improvements to C++ and Python code generation.
- Changes that apply only to Python:

- \* Improved handling of out-parameters:

- If a function or method has out-parameters we now systematically return a Python tuple containing any return value followed by the out-parameters.
- Don't treat `FILE*` or pointer-to-const as an out-parameter.

- \* Added methods for getting the content of a `mupdf.Buffer` as a Python `bytes` instance.

- \* Added Python access to nested unions in `fz_stext_block` wrapper class `mupdf.StextBlock`.

- \* Allow the MuPDF Python bindings to be installed with `pip`.

- This uses a source distribution of `mupdf` that has been uploaded to `pypi.org` in the normal way.
- Installation involves compiling the C, C++ and Python bindings so will take a few minutes. It requires `SWIG` to be installed.
- Pre-built wheels are not currently provided.

- \* Write generated C++ information into Python pickle files to allow building on systems without `clang-python`.

- \* Various changes to allow building in Python "Manylinux" containers.

- \* Allow Python access to nested unions in `fz_stext_block` wrapper. `SWIG` doesn't handle nested unions so instead we provide accessor methods in our generated C++ class.

- \* Added accessors to `fz_image`'s wrapper class.

- \* Improved generated accessor methods - e.g. ignore functions and function pointers and return `int` instead of `int8_t` to avoid SWIG getting confused.

- **2020-10-07:** Experimental release of C++ and Python bindings in MuPDF-1.18.0.

## 5.7 Building the C++, Python and C# MuPDF APIs from source

### 5.7.1 Special case for building Python bindings using pip

Python bindings can be built from source and installed into a `venv` on all platforms by using `pip`:

```
# Windows create+enter venv.
py -m venv pylocal
.\pylocal\Scripts\activate

# Unix create+enter venv.
python3 -m venv pylocal
. pylocal/bin/activate

# Upgrade pip then build and install into current venv.
python -m pip --upgrade pip
cd mupdf && python -m pip install .
```

### 5.7.2 General requirements

- Linux, Windows, or OpenBSD.
- Python development libraries.
- Build inside a Python `venv`.
- Python package `libclang` - a Python interface onto the `libclang` C/C++ parser.
  - These instructions generally use Python’s `pip` to install pypi.org’s `libclang`.
  - Other python/clang packages are available (for example pypi.org’s `clang`, or Debian’s `python-clang`) but often require explicit setting of `LD_LIBRARY_PATH` to point to the correct `libclang` dynamic library.
- `SWIG` version 3 or 4.
  - If only building the Python bindings, one can use pypi.org’s `SWIG`, installing with `python -m pip install swig`. But for simplicity the instructions below will specify a generic `SWIG` that will also generate C# bindings.
- For C# on Unix, we also need `Mono`.

## Setting up on Windows

Install Python:

- Use the Python Windows installer from the python.org website: <http://www.python.org/downloads>
- Don't use other installers such as the Microsoft Store Python package.
  - If Microsoft Store Python is already installed, leave it in place and install from python.org on top of it - uninstalling before running the python.org installer has been known to cause problems.
- A default installation is sufficient.
- Debug binaries are required for debug builds of the MuPDF Python API.
- If “Customize Installation” is chosen, make sure to include “py launcher” so that the py command will be available.
- Also see: <https://docs.python.org/3/using/windows.html>

Create and enter a Python [venv](#), upgrade to latest pip and install libclang:

```
py -m venv pylocal
.\pylocal\Scripts\activate
python -m pip install --upgrade pip
python -m pip install libclang
```

Specifying [SWIG](#):

- When running `scripts/mupdfwrap.py`, specify `--swig-windows-auto`. This will automatically download SWIG to the local directory (if not already present), and use it directly.

Specifying location of `devenv.com`:

- `scripts/mupdfwrap.py` looks for `devenv.com` in some hard-coded locations, which can be overridden with:

```
scripts/mupdfwrap.py -b --devenv <devenv.com-location> ...
```

## Setting up on Linux

Create and enter a Python [venv](#), upgrade to latest pip and install libclang:

```
python3 -m venv pylocal
. pylocal/bin/activate
python -m pip install --upgrade pip
pip install libclang
```

Install Python development libraries and [SWIG](#) using the system package manager:

```
sudo apt install python3-dev swig
```

If building the C# bindings, install [Mono](#) using the system package manager:

```
sudo apt install mono-devel
```

## Setting up on OpenBSD

Install Python (includes development libraries), [SWIG](#) and clang-python using the system package manager:

```
sudo pkg_add python py3-llvm swig
```

If building the C# bindings, install [Mono](#) using the system package manager:

```
sudo pkg_add mono
```

Notes:

- libclang and SWIG cannot be installed with pip on OpenBSD - wheels are not available and building from source fails. However unlike on other platforms, the system python-clang package py3-llvm is already integrated with the libclang shared library so it can be used directly.
- The use of the system py3-llvm package means that it is not necessary to use a Python venv on OpenBSD.

### 5.7.3 Doing a build

Get the MuPDF source tree:

```
git clone --recursive git://git.ghostscript.com/mupdf.git
```

Build C++, Python and C# bindings and run tests:

[--swig-windows-auto is only required on Windows; it will be ignored on other platforms.]

```
cd mupdf
./scripts/mupdfwrap.py --swig-windows-auto -b all --test-python --test-python-gui
./scripts/mupdfwrap.py --swig-windows-auto -b --csharp all --test-csharp --test-csharp-
→gui
```

As above but do a debug build (this requires debug version of the Python interpreter, for example python311\_d.lib):

```
./scripts/mupdfwrap.py --swig-windows-auto -d build/shared-debug -b all --test-python
```

## Notes

C# build failure: cstring.i not implemented for this target and/or Unknown directive '%cstring\_output\_allocate'.

- This is probably because SWIG is from pypi.org (e.g with `pip install swig`) and does not include support for C#.
- Solution: install SWIG using the system package manager (e.g. `sudo apt install swig` on Linux, or use `./scripts/mupdfwrap.py --swig-windows-auto ...` on Windows).

## More information

- Run `./scripts/mupdfwrap.py -h`.
- Read the doc-string at beginning of `scripts/wrap/__main__.py`.

## 5.7.4 How building the APIs works

### Building the MuPDF shared library

- On Unix, runs `make` on MuPDF's Makefile.
- On Windows, runs `devenv.com` on `.sln` and `.vcxproj` files within MuPDF's `platform/win32/` directory.

### Generation of the C++ MuPDF API

- Uses `clang-python` to parse MuPDF's C API.
- Generates C++ code that wraps the basic C interface, converting MuPDF `setjmp()/longjmp()` exceptions into C++ exceptions and automatically handling `fz_context`'s internally.
- Generates C++ wrapper classes for each `fz_*` and `pdf_*` struct, and uses various heuristics to define constructors, methods and static methods that call `fz_*`() and `pdf_*`() functions. These classes' constructors and destructors automatically handle reference counting so class instances can be copied arbitrarily.
- C header file comments are copied into the generated C++ header files.
- Compile and link the generated C++ code to create shared libraries.

### Generation of the Python and C# MuPDF APIs

- Uses SWIG to parse the previously-generated C++ headers and generate C++, Python and C# code.
- Defines some custom-written Python and C# functions and methods, e.g. so that out-params are returned as tuples.
- If SWIG is version 4+, C++ comments are converted into Python doc-comments.
- Compile and link the SWIG-generated C++ code to create shared libraries.

## 5.7.5 Building auto-generated MuPDF API documentation

Build HTML documentation for the C, C++ and Python APIs (using Doxygen and pydoc):

```
./scripts/mupdfwrap.py --doc all
```

This will generate the following tree:

```
mupdf/docs/generated/  
  index.html  
  c/  
  c++/  
  python/
```

All content is ultimately generated from the MuPDF C header file comments.

As of 2022-2-5, it looks like `swig -doxygen (swig-4.02)` ignores single-line `/** ... */` comments, so the generated Python code (and hence also Pydoc documentation) is missing information.

## 5.7.6 Generated files

File required at runtime are created in `mupdf/build/shared-<build>/`.

Other intermediate generated files are created in `mupdf/platform/`

### Details

```
mupdf/
  build/
    shared-release/    [Unix runtime files.]
      libmupdf.so      [MuPDF C API.]
      libmupdfcpp.so   [MuPDF C++ API.]
      mupdf.py         [MuPDF Python API.]
      _mupdf.so        [MuPDF Python API internals.]
      mupdf.cs         [MuPDF C# API.]
      mupdfcsharp.so   [MuPDF C# API internals.]

    shared-debug/
      [as shared-release but debug build.]

    shared-release-x32-py3.8/ [Windows runtime files.]
      mupdfcpp.dll       [MuPDF C and C++ API.]
      mupdf.py           [MuPDF Python API.]
      _mupdf.pyd         [MuPDF Python API internals.]
      mupdf.cs           [MuPDF C# API.]
      mupdfcsharp.dll    [MuPDF C# API internals.]

  platform/
    c++/
      include/          [MuPDF C++ API header files.]
        mupdf/
          classes.h
          classes2.h
          exceptions.h
          functions.h
          internal.h

      implementation/   [MuPDF C++ implementation source files.]
        classes.cpp
        classes2.cpp
        exceptions.cpp
        functions.cpp
        internal.cpp

      generated.pickle   [Information from clang parse step, used by later stages.
↪]

      windows_mupdf.def  [List of MuPDF public global data, used when linking.
↪mupdfcpp.dll.]
```

(continues on next page)

(continued from previous page)

```
python/ [SWIG Python input/output files.]
    mupdfcpp_swig.cpp
    mupdfcpp_swig.i

csharp/ [SWIG C# input/output files.]
    mupdf.cs
    mupdfcpp_swig.cpp
    mupdfcpp_swig.i

win32/
  Release/ [Windows 32-bit .dll, .lib, .exp, .pdb etc.]
  x64/
    Release/ [Windows 64-bit .dll, .lib, .exp, .pdb etc.]
      mupdfcpp64.dll
      mupdfcpp64.lib
      mupdfpyswig.dll
      mupdfpyswig.lib
```

## 5.8 Windows-specifics

### 5.8.1 Required predefined macros

Code that will use the MuPDF DLL must be built with `FZ_DLL_CLIENT` predefined.

The MuPDF DLL itself is built with `FZ_DLL` predefined.

### 5.8.2 DLLs

There is no separate C library, instead the C and C++ APIs are both in `mupdfcpp.dll`, which is built by running `devenv on platform/win32/mupdf.sln`.

The Python SWIG library is called `_mupdf.pyd` which, despite the name, is a standard Windows DLL, built from `platform/python/mupdfcpp_swig.cpp`.

### 5.8.3 DLL export of functions and data

On Windows, `include/mupdf/fitz/export.h` defines `FZ_FUNCTION` and `FZ_DATA` to `__declspec(dllexport)` and/or `__declspec(dllimport)` depending on whether `FZ_DLL` or `FZ_DLL_CLIENT` are defined.

All MuPDF C headers prefix declarations of public global data with `FZ_DATA`.

In generated C++ code:

- Data declarations and definitions are prefixed with `FZ_DATA`.
- Function declarations and definitions are prefixed with `FZ_FUNCTION`.
- Class method declarations and definitions are prefixed with `FZ_FUNCTION`.



When building `mupdfcpp.dll` on Windows we link with the auto-generated `platform/c++/windows_mupdf.def` file; this lists all C public global data.

For reasons that are not fully understood, we don't seem to need to tag C functions with `FZ_FUNCTION`, but this is required for C++ functions otherwise we get unresolved symbols when building MuPDF client code.

## 5.8.4 Building the DLLs

We build Windows binaries by running `devenv.com` directly. As of 2021-05-17 the location of `devenv.com` is hard-coded in this Python script.

Building `_mupdf.pyd` is tricky because it needs to be built with a specific `Python.h` and linked with a specific `python.lib`. This is done by setting environmental variables `MUPDF_PYTHON_INCLUDE_PATH` and `MUPDF_PYTHON_LIBRARY_PATH` when running `devenv.com`, which are referenced by `platform/win32/mupdfpyswig.vcxproj`. Thus one cannot easily build `_mupdf.pyd` directly from the Visual Studio GUI.

[In the git history there is code that builds `_mupdf.pyd` by running the Windows compiler and linker `cl.exe` and `link.exe` directly, which avoids the complications of going via `devenv`, at the expense of needing to know where `cl.exe` and `link.exe` are.]

## 5.9 C++ bindings details

### 5.9.1 Wrapper functions

Wrappers for a MuPDF function `fz_foo()` are available in multiple forms:

- Functions in the `mupdf` namespace.
  - `mupdf::ll_fz_foo()`
    - \* Low-level wrapper:
      - Does not take `fz_context*` arg.
      - Translates MuPDF exceptions into C++ exceptions.
      - Takes/returns pointers to MuPDF structs.
      - Code that uses these functions will need to make explicit calls to `fz_keep_*`() and `fz_drop_*`() .
  - `mupdf::fz_foo()`
    - \* High-level class-aware wrapper:
      - Does not take `fz_context*` arg.
      - Translates MuPDF exceptions into C++ exceptions.
      - Takes references to C++ wrapper class instances instead of pointers to MuPDF structs.
      - Where applicable, returns C++ wrapper class instances instead of pointers to MuPDF structs.
      - Code that uses these functions does not need to call `fz_keep_*`() and `fz_drop_*`() - C++ wrapper class instances take care of reference counting internally.
- Class methods
  - Where `fz_foo()` has a first arg (ignoring any `fz_context*` arg) that takes a pointer to a MuPDF struct `foo_bar`, it is generally available as a member function of the wrapper class `mupdf::FooBar`:
    - \* `mupdf::FooBar::fz_foo()`

- Apart from being a member function, this is identical to class-aware wrapper `mupdf::fz_foo()`, for example taking references to wrapper classes instead of pointers to MuPDF structs.

## 5.9.2 Constructors using MuPDF functions

Wrapper class constructors are created for each MuPDF function that returns an instance of a MuPDF struct.

Sometimes two such functions do not have different arg types so C++ overloading cannot distinguish between them as constructors (because C++ constructors do not have names).

We cope with this in two ways:

- Create a static method that returns a new instance of the wrapper class by value.
  - This is not possible if the underlying MuPDF struct is not copyable - i.e. not reference counted and not POD.
- Define an enum within the wrapper class, and provide a constructor that takes an instance of this enum to specify which MuPDF function to use.

## 5.9.3 Default constructors

All wrapper classes have a default constructor.

- For POD classes each member is set to a default value with `this->foo = {};`. Arrays are initialised by setting all bytes to zero using `memset()`.
- For non-POD classes, class member `m_internal` is set to `nullptr`.
- Some classes' default constructors are customized, for example:
  - The default constructor for `fz_color_params` wrapper `mupdf::FzColorParams` sets state to a copy of `fz_default_color_params`.
  - The default constructor for `fz_md5` wrapper `mupdf::FzMd5` sets state using `fz_md5_init()`.
  - These are described in class definition comments in `platform/c++/include/mupdf/classes.h`.

## 5.9.4 Raw constructors

Many wrapper classes have constructors that take a pointer to the underlying MuPDF C struct. These are usually for internal use only. They do not call `fz_keep_*`() - it is expected that any supplied MuPDF struct is already owned.

## 5.9.5 POD wrapper classes

Class wrappers for MuPDF structs default to having a `m_internal` member which points to an instance of the wrapped struct. This works well for MuPDF structs which support reference counting, because we can automatically create copy constructors, `operator=` functions and destructors that call the associated `fz_keep_*`() and `fz_drop_*`() functions.

However where a MuPDF struct does not support reference counting and contains simple data, it is not safe to copy a pointer to the struct, so the class wrapper will be a POD class. This is done in one of two ways:

- `m_internal` is an instance of the MuPDF struct, not a pointer.
  - Sometimes we provide members that give direct access to fields in `m_internal`.
- An 'inline' POD - there is no `m_internal` member; instead the wrapper class contains the same members as the MuPDF struct. This can be a little more convenient to use.

### 5.9.6 Extra static methods

Where relevant, wrapper class can have static methods that wrap selected MuPDF functions. For example `FzMatrix` does this for `fz_concat()`, `fz_scale()` etc, because these return the result by value rather than modifying a `fz_matrix` instance.

### 5.9.7 Miscellaneous custom wrapper classes

The wrapper for `fz_outline_item` does not contain a `fz_outline_item` by value or pointer. Instead it defines C++-style member equivalents to `fz_outline_item`'s fields, to simplify usage from C++ and Python/C#.

The fields are initialised from a `fz_outline_item` when the wrapper class is constructed. In this particular case there is no need to hold on to a `fz_outline_item`, and the use of `std::string` ensures that value semantics can work.

## 5.10 Python/C# bindings details

### 5.10.1 Python differences from C API

[The functions described below are also available as class methods.]

#### Custom methods

Python and C# code does not easily handle functions that return raw data, for example as an `unsigned char*` that is not a zero-terminated string. Sometimes we provide a C++ method that returns a `std::vector` by value, so that Python and C# code can wrap it in a systematic way.

For example `Md5::fz_md5_final2()`.

#### New functions

- `fz_buffer_extract_copy()`: Returns copy of buffer data as a Python bytes.
- `fz_buffer_storage_memoryview()`: Returns Python `memoryview` onto buffer data. Relies on buffer contents not changing.
- `fz_pixmap_samples2()`: Returns Python `memoryview` onto `fz_pixmap` data.

#### Implemented in Python

- `fz_format_output_path()`
- `pdf_dict_getl()`
- `pdf_dict_putl()`

## Non-standard API or implementation

- `fz_buffer_extract()`: Returns a *copy* of the original buffer data as a Python bytes. Still clears the buffer.
- `fz_convert_color()`: No `float* fv` param, instead returns `(rgb0, rgb1, rgb2, rgb3)`.
- `fz_fill_text()`: `color` arg is tuple/list of 1-4 floats.
- `fz_new_buffer_from_copied_data()`: Takes Python bytes instance.
- `fz_set_error_callback()`: Takes a Python callable; no `void* user` arg.
- `fz_set_warning_callback()`: Takes a Python callable; no `void* user` arg.
- `fz_warn()`: Takes single Python str arg.
- `ll_fz_convert_color()`: No `float* fv` param, instead returns `(rgb0, rgb1, rgb2, rgb3)`.
- `ll_pdf_set_annot_color()`: Takes single color arg which must be float or tuple of 1-4 floats.
- `ll_pdf_set_annot_interior_color()`: Takes single color arg which must be float or tuple of 1-4 floats.
- `pdf_dict_putl_drop()`: Always raises exception because not useful with automatic ref-counts.
- `pdf_field_name()`: Uses extra C++ function `pdf_field_name2()` which returns `std::string` by value.
- `pdf_set_annot_color()`: Takes single color arg which must be float or tuple of 1-4 floats.
- `pdf_set_annot_interior_color()`: Takes single color arg which must be float or tuple of 1-4 floats.

## 5.10.2 Making MuPDF function pointers call Python code

### Overview

For MuPDF structs with function pointers, we provide a second C++ wrapper class for use by the Python bindings.

- The second wrapper class has a 2 suffix, for example `PdfFilterOptions2`.
- This second wrapper class has a virtual method for each function pointer, so it can be used as a [SWIG Director class](#).
- Overriding a virtual method in Python results in the Python method being called when MuPDF C code calls the corresponding function pointer.
- One needs to activate the use of a Python method as a callback by calling the special method `use_virtual_<method-name>()`. [It might be possible in future to remove the need to do this.]
- It may be possible to use similar techniques in C# but this has not been tried.

### Callback args

Python callbacks have args that are more low-level than in the rest of the Python API:

- Callbacks generally have a first arg that is a SWIG representation of a MuPDF `fz_context*`.
- Where the underlying MuPDF function pointer has an arg that is a pointer to an MuPDF struct, unlike elsewhere in the MuPDF bindings we do not translate this into an instance of the corresponding wrapper class. Instead Python callbacks will see a SWIG representation of the low-level C pointer.
  - It is not safe to construct a Python wrapper class instance directly from such a SWIG representation of a C pointer, because it will break MuPDF's reference counting - Python/C++ constructors that take a raw pointer to a MuPDF struct do not call `fz_keep_*`() but the corresponding Python/C++ destructor will call `fz_drop_*`()

- It might be safe to create an wrapper class instance using an explicit call to `mupdf.fz_keep_*`, but this has not been tried.
- As of 2023-02-03, exceptions from Python callbacks are propagated back through the Python, C++, C, C++ and Python layers. The resulting Python exception will have the original exception text, but the original Python backtrace is lost.

## Example

Here is an example PDF filter written in Python that removes alternating items:

### Details

```
import mupdf

def test_filter(path):
    class MyFilter( mupdf.PdfFilterOptions2):
        def __init__( self):
            super().__init__()
            self.use_virtual_text_filter()
            self.recurse = 1
            self.sanitize = 1
            self.state = 1
            self.ascii = True
        def text_filter( self, ctx, ucsbuf, ucslen, trm, ctm, bbox):
            print( f'text_filter(): ctx={ctx} ucsbuf={ucsbuf} ucslen={ucslen} trm={trm}
↪ ctm={ctm} bbox={bbox}')
            # Remove every other item.
            self.state = 1 - self.state
            return self.state

    filter_ = MyFilter()

    document = mupdf.PdfDocument(path)
    for p in range(document.pdf_count_pages()):
        page = document.pdf_load_page(p)
        print( f'Running document.pdf_filter_page_contents on page {p}')
        document.pdf_begin_operation('test filter')
        document.pdf_filter_page_contents(page, filter_)
        document.pdf_end_operation()

    document.pdf_save_document('foo.pdf', mupdf.PdfWriteOptions())
```



## PROGRESSIVE LOADING

### 6.1 What is progressive loading?

The idea of progressive loading is that as you download a *PDF* file into a browser, you can display the pages as they appear.

*MuPDF* can make use of 2 different mechanisms to achieve this. The first relies on the file being “linearized”, the second relies on the caller of *MuPDF* having fine control over the http fetch and on the server supporting byte-range fetches.

For optimum performance a file should be both linearized and be available over a byte-range supporting link, but benefits can still be had with either one of these alone.

### 6.2 Progressive download using “linearized” files

*Adobe* defines “linearized” PDFs as being ones that have both a specific layout of objects and a small amount of extra information to help avoid seeking within a file. The stated aim is to deliver the first page of a document in advance of the whole document downloading, whereupon subsequent pages will become available. *Adobe* also refers to these as “Optimized for fast web view” or “Web Optimized”.

In fact, the standard outlines (poorly) a mechanism by which ‘hints’ can be included that enable the subsequent pages to be found within the file too. Unfortunately this is very poorly supported with many tools, and so the hints have to be treated with suspicion.

*MuPDF* will attempt to use hints if they are available, but will also use a linear search of the file to discover pages if not. This means that the first page will be displayed quickly, and then subsequent ones will appear with ‘incomplete’ renderings that improve over time as more and more resources are gradually delivered.

Essentially the file starts with a slightly modified header, and the first object in the file is a special one (the linearization object) that:

- indicates that the file is linearized.
- gives some useful information (like the number of pages in the file).

This object is then followed by all the objects required for the first page, then the “hint stream”, then sets of object for each subsequent page in turn, then shared objects required for those pages, then various other random things.

---

**Note:** While page 1 is sent with all the objects that it uses, shared or otherwise, subsequent pages do not get shared resources until after all the unshared page objects have been sent.

---

## 6.3 The Hint Stream

*Adobe* intended “Hint Stream” to be useful to facilitate the display of subsequent pages, but it has never used it. Consequently you can’t trust people to write it properly - indeed *Adobe* outputs something that doesn’t quite conform to the specification.

Consequently very few people actually use it. *MuPDF* will use it after sanity checking the values, and should cope with illegal/incorrect values.

## 6.4 So how does *MuPDF* handle progressive loading?

*MuPDF* has made various extensions to its mechanisms for handling progressive loading.

### 6.4.1 Progressive streams

At its lowest level *MuPDF* reads file data from a `fz_stream`, using the `fz_open_document_with_stream` call. (`fz_open_document` is implemented by calling this). We have extended the `fz_stream` slightly, giving the system a way to ask for meta information (or perform meta operations) on a stream.

Using this mechanism *MuPDF* can query:

- whether a stream is progressive or not (i.e. whether the entire stream is accessible immediately).
- what the length of a stream should ultimately be (which an http fetcher should know from the Content-Length header).

With this information *MuPDF* can decide whether to use its normal object reading code, or whether to make use of a linearized object. Knowing the length enables us to check with the length value given in the linearized object - if these differ, the assumption is that an incremental save has taken place, thus the file is no longer linearized.

When data is pulled from a progressive stream, if we attempt to read data that is not currently available, the stream should throw a `FZ_ERROR_TRYLATER` error. This particular error code will be interpreted by the caller as an indication that it should retry the parsing of the current objects at a later time.

When a *MuPDF* call is made on a progressive stream, such as `fz_open_document_with_stream`, or `fz_load_page`, the caller should be prepared to handle a `FZ_ERROR_TRYLATER` error as meaning that more data is required before it can continue. No indication is directly given as to exactly how much more data is required, but as the caller will be implementing the progressive `fz_stream` that it has passed into *MuPDF* to start with, it can reasonably be expected to figure out an estimate for itself.

### 6.4.2 Cookie

Once a page has been loaded, if its contents are to be ‘run’ as normal (using e.g. `fz_run_page`) any error (such as failing to read a font, or an image, or even a content stream belonging to the page) will result in a rendering that aborts with an `FZ_ERROR_TRYLATER` error. The caller can catch this and display a placeholder instead.

If each page’s data was entirely self-contained and sent in sequence this would perhaps be acceptable, with each page appearing one after the other. Unfortunately, the linearization procedure as laid down by *Adobe* does **not** do this: objects shared between multiple pages (other than the first) are not sent with the pages themselves, but rather **after** all the pages have been sent.



This means that a document that has a title page, then contents that share a font used on pages 2 onwards, will not be able to correctly display page 2 until after the font has arrived in the file, which will not be until all the page data has been sent.

To mitigate against this, *MuPDF* provides a way whereby callers can indicate that they are prepared to accept an ‘incomplete’ rendering of the file (perhaps with missing images, or with substitute fonts).

Callers prepared to tolerate such renderings should set the `incomplete_ok` flag in the cookie, then call `fz_run_page` etc. as normal. If a `FZ_ERROR_TRYLATER` error is thrown at any point during the page rendering, the error will be swallowed, the ‘incomplete’ field in the cookie will become non-zero and rendering will continue. When control returns to the caller the caller can check the value of the ‘incomplete’ field and know that the rendering it received is not authoritative.

## 6.5 Progressive loading using byte range requests

If the caller has control over the *http* fetch, then it is possible to use byte range requests to fetch the document ‘out of order’. This enables non-linearized files to be progressively displayed as they download, and fetches complete renderings of pages earlier than would otherwise be the case. This process requires no changes within *MuPDF* itself, but rather in the way the progressive stream learns from the attempts *MuPDF* makes to fetch data.

Consider, for example, an attempt to fetch a hypothetical file from a server.

- The initial http request for the document is sent with a “Range:” header to pull down the first (say) 4k of the file.
- As soon as we get the header in from this initial request, we can respond to meta stream operations to give the length, and whether byte requests are accepted.
  - If the header indicates that byte ranges are acceptable the stream proceeds to go into a loop fetching chunks of the file at a time (not necessarily in-order). Otherwise the server will ignore the Range: header, and just serve the whole file.
  - If the header indicates a content-length, the stream returns that.
- *MuPDF* can then decide how to proceed based upon these flags and whether the file is linearized or not. (If the file contains a linearized object, and the content length matches, then the file is considered to be linear, otherwise it is not).

**If the file is linear:**

- We proceed to read objects out of the file as it downloads. This will provide us the first page and all its resources. It will also enable us to read the hint streams (if present).
- Once we have read the hint streams, we unpack (and sanity check) them to give us a map of where in the file each object is predicted to live, and which objects are required for each page. If any of these values are out of range, we treat the file as if there were no hint streams.
- If we have hints, any attempt to load a subsequent page will cause *MuPDF* to attempt to read exactly the objects required. This will cause a sequence of seeks in the `fz_stream` followed by reads. If the stream does not have the data to satisfy that request yet, the stream code should remember the location that was fetched (and fetch that block in the background so that future retries will succeed) and should raise an `FZ_ERROR_TRYLATER` error.

---

**Note:** Typically therefore when we jump to a page in a linear file on a byte request capable link, we will quickly see a rough rendering, which will improve fairly fast as images and fonts arrive.

---

- Regardless of whether we have hints or byte requests, on every `fz_load_page` call *MuPDF* will attempt to process more of the stream (that is assumed to be being downloaded in the background).

As linearized files are guaranteed to have pages in order, pages will gradually become available. In the absence of byte requests and hints however, we have no way of getting resources early, so the renderings for these pages will remain incomplete until much more of the file has arrived.

---

**Note:** Typically therefore when we jump to a page in a linear file on a non byte request capable link, we will see a rough rendering for that page as soon as data arrives for it (which will typically take much longer than would be the case with byte range capable downloads), and that will improve much more slowly as images and fonts may not appear until almost the whole file has arrived.

---

- When the whole file has arrived, then we will attempt to read the outlines for the file.

**For a non-linearized PDF on a byte request capable stream:**

- *MuPDF* will immediately seek to the end of the file to attempt to read the trailer. This will fail with a `FZ_ERROR_TRYLATER` due to the data not being here yet, but the stream code should remember that this data is required and it should be prioritized in the background fetch process.
- Repeated attempts to open the stream should eventually succeed therefore. As *MuPDF* jumps through the file trying to read first the xrefs, then the page tree objects, then the page contents themselves etc., the background fetching process will be driven by the attempts to read the file in the foreground.

---

**Note:** Typically therefore the opening of a non-linearized file will be slower than a linearized one, as the xrefs/page trees for a non-linear file can be 20%+ of the file data. Once past this initial point however, pages and data can be pulled from the file almost as fast as with a linearized file.

---

**For a non-linearized PDF on a non-byte request capable stream:**

- *MuPDF* will immediately seek to the end of the file to attempt to read the trailer. This will fail with a `FZ_ERROR_TRYLATER` due to the data not being here yet. Subsequent retries will continue to fail until the whole file has arrived, whereupon the whole file will be instantly available.

---

**Note:** This is the worst case situation - nothing at all can be displayed until the entire file has downloaded.

---

A typical structure for a fetcher process (see `curl-stream.c`, `mupdf-curl` in `platform/win32/mupdf-curl.vcxproj`) as an example) might therefore look like this:

- We consider the file as an (initially empty) buffer which we are filling by making requests. In order to ensure that we make maximum use of our download link, we ensure that whenever one request finishes, we immediately launch another. Further, to avoid the overheads for the request/response headers being too large, we may want to divide the file into ‘chunks’, perhaps 4 or 32k in size.
- We can then have a receiver process that sits there in a loop requesting chunks to fill this buffer. In the absence of any other impetus the receiver should request the next ‘chunk’ of data from the file that it does not yet have, following the last fill point. Initially we start the fill point at the beginning of the file, but this will move around based on the requests made of the progressive stream.
- Whenever *MuPDF* attempts to read from the stream, we check to see if we have data for this area of the file already. If we do, we can return it. If not, we remember this as the next “fill point” for our receiver process and throw a `FZ_ERROR_TRYLATER` error.

## CHANGES

- For the full list of changes between versions of *MuPDF* see [CHANGES](#).
- For changes to *existing APIs* see below.

### 7.1 Changes to the existing *API*

#### Why does the *API* change?

From time to time, during development, it becomes clear that the public *API* needs to change; either because the existing *API* has flaws in it, or to accommodate new functionality. Such changes are not undertaken lightly, and they are kept as minimal as possible.

The alternative would be to freeze the *API* and to introduce more and more compatibility veneers, ultimately leading to a bloated *API* and additional complexity. We have done this in the past, and will do it again in future if circumstances demand it, but we consider small changes in the *API* to be a price worth paying for clarity and simplicity.

To minimise the impact of such changes, we undertake to list the *API* changes between different versions of the library here, to make it as simple as possible for integrators to make the small changes that may be require to update between versions.

---

**Important:** The changes listed below only affects *existing APIs*.

---

#### 7.1.1 Changes from 1.20 to 1.21

None.

#### 7.1.2 Changes from 1.19 to 1.20

Text search *API* extended to be able to distinguish between separate search hits.

### 7.1.3 Changes from 1.18 to 1.19

We were inconsistent with the behaviour of `fz_create_` and `pdf_create_` functions, in that sometimes they returned a borrowed reference, rather than a reference that the caller had to drop. We now always return a reference that the caller owns and must drop.

### 7.1.4 Changes from 1.17 to 1.18

None.

### 7.1.5 Changes from 1.16 to 1.17

The accessors for reading and creating `QuadPoints`, `InkList` and `Vertices` data for *Highlight*, *Underline*, *StrikeOut*, *Squiggle*, *Ink*, *Polygon*, and *PolyLine* annotation types have been simplified. They now take and return `fz_quad` and `fz_point` structs instead of float arrays. We have also added functions to construct the datastructures one piece at a time, removing the need to allocate a temporary array to pass.

- `typedef struct fz_quad { fz_point ul, ur, ll, lr; };`

To facilitate loading *EPUB* documents without laying out the entire document at once, we have introduced chapters into the document structure. Each document instead of having just a plain list of pages, now has a list of chapters, and each chapter has a list of pages. Most of the old functions have the same functionality, but we have added several new functions, which if used, will provide significant speedups when jumping to a random page in large *EPUB* documents.

- `int fz_count_chapters(ctx, doc)`
- `int fz_count_chapter_pages(ctx, doc, chapter)`
- `fz_page *fz_load_chapter_page(ctx, doc, chapter, page)`
- `int fz_search_chapter_page_number(ctx, doc, chapter, page, needle, hit_quads, hit_size)`

However, in order to support bookmarks and links using chapters we have had to introduce a new struct `fz_location`, and the functions to resolve links have changed to use this new struct.

- `typedef struct { int chapter, page; } fz_location;`
- `fz_location fz_resolve_link(ctx, doc, uri, x, y);`

Functions to map between a fixed page number and a `fz_location` have been added to help facilitate migration to the new *API*.

- `int fz_page_number_from_location(ctx, doc, location);`
- `fz_location fz_location_from_page_number(ctx, doc, number);`
- `fz_location fz_next_page(ctx, doc, location);`
- `fz_location fz_previous_page(ctx, doc, location);`
- `fz_location fz_last_page(ctx, doc);`

The layout information for each chapter can at your option be cached in an “accelerator” file for even faster loading. This will help performance when using the old page number based rather than location based functions.

- `fz_document fz_open_accelerated_document(ctx, filename, accelerator_filename);`
- `void fz_save_accelerator(ctx, document, accelerator_filename);`

## 7.1.6 Changes from 1.15 to 1.16

There has been a major overhaul of the color management architecture. Unless you're implementing custom devices or interpreters, this should only have a minor impact.

- Argument order when passing color and colorspace to functions regularized: `sourceColorspace`, `sourceColorArray`, `destinationColorspace`, `destinationColorArray`, `proofColorspace`, `colorParams`.
- Pass `fz_color_params` argument by value.
- Changed `fz_default_parameters` from a function to a global constant.
- Replaced `fz_set_icc_engine` with `fz_enable_icc` and `fz_disable_icc` to toggle color management at runtime.
- Replaced pixmap color converter struct with a single `fz_convert_pixmap` function call.
- Replaced `fz_cal_colorspace` struct with constructor to create an ICC-backed calibrated colorspace directly.
- Passing `NULL` is not a shortcut for *DeviceGray* any more!
- Added public definitions for *Indexed* and *Separation* colorspaces.
- Changed colorspace constructors.

The `fz_set_stdout` and `fz_set_stderr` functions have been removed. If you were using these to capture warning and error messages, use the new user callbacks for warning and error logging instead: `fz_set_warning_callback` and `fz_set_error_callback`.

The structured text *html* and *xhtml* output formats take an additional argument: the page number. This number is used to create an *id* attribute for each page to use as a hyperlink target.

## 7.1.7 Changes from 1.14 to 1.15

- **PDF Portfolios**

This functionality has been removed. We do not believe anyone was using this. If you were, please contact us for assistance. This functionality can be achieved using “`mutool run`” and `docs/examples/pdf-portfolio.js`.

- **FZ\_ERROR\_TRYLATER**

This functionality has been removed. We do not believe anyone was using this. If you were, please contact us for assistance.

- **Annotations/Forms**

We are undertaking a significant rework of this functionality at the moment. We do not believe anyone is using this at the moment, and would therefore encourage anyone who is to contact us for help in upgrading.

- **Various functions involving `fz_colorspace` have lost consts.**

`fz_colorspaces` are immutable once created, other than changes due to reference counting. Passing a const `fz_colorspace` to a function that might keep a reference to it either has to take a non const `fz_colorspace` pointer, or take a const one, and ‘break’ the const. Having some functions take const `fz_colorspace` and some not is confusing, so therefore, for simplicity, all `fz_colorspaces` are now passed as non const. This should not affect any user code.

- **`fz_process_shade()`**

This now takes an extra ‘scissor’ argument. To upgrade old code, if you don’t have an appropriate scissor rect available, it is safe (but unwise) to pass `fz_infinite_rect`.

- **`fz_tint_pixmap()`**

Rather than taking `r`, `g` and `b`, values to tint with, the function now takes 8 bit hex rgb values for black and white, enabling greater control, allowing “low contrast” and “night view” effects.

- **pdf\_add\_image()**  
This no longer requires a mask flag. The image already knows if it is a mask.
- **pdf\_processor.op\_BI()**  
The op\_BI callback is now passed an additional colorspace resource name.

## THIRD PARTY LIBRARIES USED BY *MUPDF*

These are the third party libraries used by *MuPDF*.

Library	Version	Function	License
<b>Required</b>			
freetype	2.12.1	Font scaling and rendering	BSD-style
harfbuzz	5.3.0	Text shaping	MIT-style
libjpeg	9.0e with patches	JPEG decoding	BSD-style
Incompatible fork of lcms2	2.14 with patches	Color management	MIT-style
openjpeg	2.5.0	JPEG 2000 decoding	BSD-style
zlib	1.2.13	Deflate compression	zlib License
gumbo-parser	0.10.1	HTML5 parser	Apache 2.0
<b>Optional</b>			
FreeGLUT	3.0.0 with patches	OpenGL API for UI	MIT-style
curl	7.66.0 with patches	HTTP data transfer	MIT-style
JPEG-XR reference	1.32 with patches	JPEG-XR decoding	special
Tesseract	5.0.0-alpha-20201231 with patches	OCR	Apache 2.0
Leptonica	1.80.0 with patches	Tesseract dependency	BSD-style

---

**Note:** *jbig2dec* and *MuJS* are included in “thirdparty” but are copyright *Artifex Software Inc.*

---