libhpdftbl

Generated on Tue May 31 2022 08:31:59 for libhpdftbl by Doxygen 1.9.3

Chapter 1

Overview

1.1 What is this?

The Haru PDF library is a great way to programmatically produce PDFs from programs. However, in many instances the best way to present data produced is as a grid (or table). To manually create and setup such tables int Haru PDF library is of course possible but only painstakingly so.

This C/C++ library `libhpdftbl` will facilitate the creation of tables with the Haru PDF library as well as handling the pesky issue of character conversion needed between UTF-8 and the internal standard used by PDF and Lib Haru. In addition to mere normal table the library also supports the creation of forms where each cell has a label similar to "formal" paper forms. This is a great way to present structured data from a DB.

This library provides a flexible abstraction for creating advanced tables with a model-view-controller like setup. This allows an easy way to separate the layout of the table from the actual data in the table.

1.2 Features

- Supports both OSX/Linux builds and their different dynamic library variants
- Fully supports UTF-8 with automatic conversion to PDF character encoding
- Supports multiple paradigms for creating and populating tables
    - Directly store value in table cell
    - Create a data structure (2D-Array) with all data to be set at once
    - Use callback populating functions with identifying tags for each table cell
- Allow table to be serialized to JSON and read back
- Options ot use labels in table cell to create forms
- Row and column spanning of cells
- Support for predefined widgets in table cell to illustrate values
- Complete control of background color, fonts, and frame colors
- Possible to use table themes that provides pre-defined look-and-feel for table
- Allows theme to be serialized to JSON and read beck
- Both dynamic and static library provided
- Last but not least; extensive `documentation` which is also included in the distribution in both PDF and HTML format.

1.3 Some Examples

Note

All code examples can be found in the `examples/` directory and in the `Examples` section in the documentations.

1.3.1 Example 1 - Plain table with header

tut_ex02_1.c

Header 0	Header 1	Header 2	Header 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15

1.3.2 Example 2 - Table with cell labels

example01.c

Label 0: Content 0	Label 1: Content 1	Label 2: Content 2	Label 3: Content 3
Label 4: Content 4	Label 5: Content 5	Label 6: Content 6	Label 7: Content 7
Label 8: Content 8	Label 9: Content 9	Label 10: Content 10	Label 11: Content 11
Label 12: Content 12	Label 13: Content 13	Label 14: Content 14	Label 15: Content 15
Label 16: Content 16	Label 17: Content 17	Label 18: Content 18	Label 19: Content 19

1.3.3 Example 2 - Plain table with row/column spanning and table title

example01.c

Example 3: Table cell spannings and full grid and header

Content 0	Content 1		
Label 4: Content 4	Label 5: Content 5		
Label 8: Content 8	Label 9: Content 9	Label 10: Content 10	
Label 12: Content 12	Label 13: Content 13	Label 14: Content 14	Label 15: Content 15
Label 16: Content 16	Label 17: Content 17		
Label 20: Content 20			
Label 24: Content 24	Label 25: Content 25	Label 26: Content 26	Label 27: Content 27
Label 28: Content 28	Label 29: Content 29	Label 30: Content 30	
Label 32: Content 32	Label 33: Content 33		

1.3.4   Example 3 - Table with labels and cell widgets

example01.c

Example 5: Using widgets in cells

Horizontal seg bar:	Label 1: Content 1	Label 2: Content 2	Label 3: Content 3
Horizontal bar:	Label 5: Content 5	Label 6: Content 6	Label 7: Content 7
Slider on:	Label 9: Content 9	Label 10: Content 10	Label 11: Content 11
Slider off:	Label 13: Content 13	Label 14: Content 14	Label 15: Content 15
Strength meter:	Label 17: Content 17	Label 18: Content 18	Label 19: Content 19
Boxed letters:	Label 21: Content 21	Label 22: Content 22	Label 23: Content 23

Chapter 2

Building the library

2.1 The short version; TL; DR

For the long version see Building from source

If the necessary pre-requisites are fulfilled the distributed tar-ball can be built with:
```
$ tar xzf libhpdftbl-<version>.tar.gz
$ cd libhpdftbl-<version>
$ ./configure && make
```

If any libraries are missing the `configure` process will discover this and tell what needs to be installed. Use `./configure -h` to see a list of possible options for configuring the build.

To then verify the build run
```
$ make check
```

which will execute all the unit-tests in the library. If all tests pass a *Success!* message can bee seen at the end of all listed tests. The library can then be installed with
```
$ make install
```

If the install succeeds this will, by default, install the library in the `/usr/local` subtree, both a static and dynamic version of the library will by default be installed.

Note

> What directory to install to can be adjusted by using the `--prefix` argument when configuring the build.

2.2 Pre-requisites

There are two mandatory and one semi-otional external library required to fully build the libhpdftbl:

1. **libhpdf** - The Haru PDF library. On OSX this is most easily installed by using the `brew` OSX package manager. The library is available as `libharu` as of this writing the latest version is `libharu-2.3.0`

2. **iconv** - The character encoding conversion library. On OSX $>$ 11.x this is included by default once you have `xcode` command line tools installed which is basically a pre-requisite required for all development on OSX. *On ancient versions of OSX this was not the case.*

3. **libjansson** - Library to parse JSON files. Needed to enable table serialization to- and from JSON format. *Note:* This can be omitted but then no serialization functions will be available in the library.

2.2.1 Different versions of iconv on OSX

Note

> OSX Package manager: We recommend using `brew` as the package manager for OSX.

Unfortunately there are two different (and incompatible) versions of `libiconv` readily available for OSX. One library that uses the prefix `iconv_*` and the other `libiconv_*` on its exported functions. Compiling `libhpdftbl` requires the first of these which is the prevalent version and the default on both OSX and Linux.

This is almost exclusively an issue for those that actively develop on OSX and may have over time installed multiple versions of libraries and as such are aware of these challenges.

2.2.2 OSX native libiconv

After installing `xcode` command line tools on OSXit is safe to assume that a library called `/usr/lib/iconv.dylib` is available.

However, if one tries to list this library in `/usr/lib` there will not be a `libiconv.dylib`. Still, if the code is linked with `-liconv` it will work as expected. How come?

The reason is the way OSX handles different library versions for different OSX SDKs. Since `xcode` supports developing for different OSX versions the SDK would need to include a complete setup of all `*.dylib` of the right version for each included version of the SDK. To reduce diskspace all dynamic libraries are rolled-up in a dynamic link shared cache for each SDK version. The tool chain (e.g. `clang`) have been augmented to be aware of this. Hence, there is no need to have libraries in `/usr/lib`. Instead, OSX from v11 and onwards uses the concept of *stub libraries* with suffix `*.tbd` for each supported SDK version (tbd stands for "text based description"). They are small text files with meta information about the library used by the tool-chain.

For example for SDK 12.3 the stub for libiconv can be found at
`/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/lib/libiconv.tbd`

and the corresponding include header is located at
`/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/iconv.h`

2.2.3 OSX GNU port of libiconv

If you have happened to install `libiconv` via the MacPorts you are out of luck and need to change. MacPorts uses the GNU version which uses the prefix "`libiconv_*`" for its exported function and is not compatible since the table library assumes the naming convention of the standard OSX version (after v11)

2.2.4 Troubleshooting OSX <tt>libiconv</tt>

If the build complains about `libiconv` the following steps could be take to try to track down the problem:

1. Find out all installed versions of `libiconv` on your machine

   ```
   $> find / -iregex '.*/libiconv.*' 2> /dev/null
   ```

   The "`2> /dev/null`" makes sure you don't get a lot of noise with "permission denied"

2. Find out the SDK path that is actively used

   ```
   $> xcrun --show-sdk-path
   ```

3. Check you `PATH` variable

   ```
   $> echo $PATH
   ```

2.3 Building the library from source

There are two levels of rebuilding the library that we will discuss

1. Using a build environment to rebuild the library (i.e. building from the supplied tar-ball)

2. Rebuilding from a cloned repo and rebuild the build environment from scratch. As a principle no generated files are stored in the repo.

2.3.1 Rebuilding using av existing build environment

Rebuilding the library using a pre-configured build environment requires `gcc` (or `clang`) and `make` together with the standard C/C++ libraries to build the library.

The library source with suitable build-environment is distributed as a tar-ball

1. `libhpdftbl-x.y.z.tar.gz`

This tar-ball includes a build environment constructed with the GNU autotools. This means that after downloading the tar-ball you can rebuild the library as so:

```
$ tar xzf libhpdftbl-x.y.z.tar.gz
$ cd libhpdf-x.y.z
$ ./configure && make
... (output from the configuration and build omitted) ...
```

and then (optionally) install the library with

```
$ make install
```

By default, the library will install under the `/usr/local` but that can be adjusted by using the `--prefix` parameter to `configure`. For example

```
$ tar xzf libhpdftbl-1.0.0.tar.gz
$ cd libhpdf-1.0.0
$ ./configure --prefix=/usr && make
... (output from the configuration and build omitted) ...
```

Please refer to `configure -h` for other possible configurations. As a shortcut two utility scripts are included that give some extra `CFLAGS` flags to either compile the library for production use `./scripts/stdbld.sh` or for debugging `./scripts/dbgbld.sh` (See Some notes on Debugging)

2.3.2 Rebuilding from a cloned repo

Note

> This is for experienced developers!

The repo does not include any of the generated files as the tar-ball does. This means that the following build tools needs to be installed in order to fully rebuild from a cloned repo.

1. A complete set of GNU compiler chain (or on OSX clang)

2. GNU autotools (autoconf, automake, libtool)

3. Doxygen in order to rebuild the documentation.

Since it is completely out of the scope to describe the intricacies of the GNU autotools we will only show what to do assuming this tool chain have already been installed.

To simplify the bootstrapping necessary to create a full autotools environment from the cloned repo a utility script that does this is provided in the form of `./scripts/bootstrap.sh`. After cloning the repo run (from the `libhpdftbl` top directory)

```
$ ./scripts/bootstrap.sh
```

This script will now run `autoreconf`, `automake`, `glibtoolize` as needed in order to create a full build environment. It will also run `configure` and if everything works as expected the last lines you will see (on OSX) will be

```
...
config.status: executing libtool commands
configure: ----------------------------------------------------------------------------
configure: INSTALLATION SUMMARY:
configure:   - Build configured for OSX.
configure:   - Can rebuild HTML docs with Doxygen.
configure:   - Can also create PDF docs (have LaTeX).
configure:   - Installing to /usr/local
configure: ----------------------------------------------------------------------------
```

and then to compile the library with

```
$ make
```

The simplest way to verify that everything works is to run the built-in unit/integration tests

```
$ make check
Info: PASS: tut_ex01
Info: PASS: tut_ex02
<omitted ...>
Info: PASS: tut_ex20
Info: ================================
Info: SUCCESS! 20/20 tests passed.
Info: ================================
```

To install the library use

```
$> make install
```

This will install headers and library under `/usr/local` (unless the prefix was changed when running the `configure`)

2.4 Miscellaneous

2.4.1   Some notes on Compiling for debugging

Since the library builds with `libtool` and this tool will generate a wrapper shell script for each example to load the, not yet installed, library it also means this "executable" shell script cannot directly be used to debug with for example `gdb`.

The solution for this is to configure the library to only build static libraries which are directly linked with the example binaries and as such can be debugged as usual. It is also a good idea to disable optimization during debugging to make the source better follow the execution while stepping through the code. This configuration is done with:

```
$> ./configure --disable-shared CFLAGS="-O0 -ggdb"
```

After this all the examples will be statically linked and can be debugged as usual

An alternative way (as recommended in the [libtool manual](#)) is to launch the debugger with:

```
$> libtool --mode=execute gdb <example program>
```

This will run the `gdb` debugger from command line. For debugging from within a IDE (like Netbeans, Clion, etc.) use the static library method.

As a convenience a script is provided to handle the debug build configuration [scripts/dbgbld.sh](#)

2.4.2   Some notes on updating the documentation

By design the documentation is not updated by the default make target in order minimize the build time during development. To rebuild the *html* documentation build the target
```
$> make html
```

and to rebuild the *PDF* version build (assuming you have LaTeX installed)
```
$> make pdf
```

The resulting documentations are stored under `docs/out/html` and `docs/out/latex/refman.pdf`

Warning

> There is a shell script `scripts/docupload.sh.in` that the author (i.e. me!) uses to upload the HTML and PDF documentation to the GitHub pages of the author. For obvious reason this script will not work for anyone else since it requires write access to the doc repo (through an SSL certificate).

2.4.3   Some notes on Windows build

The source files are suitable augmented to also compile on MS Windows with selective defines. However, this is on a best effort basis since I have no longer access to a Windows system to verify the workings. So, basically there is expected to some minor configuration issues when building on windows. Patches are welcome!

2.4.4   Some notes on using C or C++ to build

The source files are also suitable augmented to compile on both a C and a C++ compiler. However, the default build environment is set up for a pure C library build. To add a configuration switch for this would be the sensible way to handle this. This is not done and again, is left as an exercise for the reader.

Chapter 3

Getting started

In this section we will introduce the basic usage of the `hpdftbl` library. We will start simple and work us all the way to complex tables and explain what is happening as we go along.

We will not assume any knowledge of the table library, but **we will assume that you are familiar with the plain Haru PDF library**.

3.1 Creating an infrastructure for the examples

Before we start creating a table we need to set up a plain PDF page with the core HPDF library. The HPDF library has excellent documentation on how to do this, and we will use the same basic setup for all our examples. We will create a document in A4 size that have one page that will be written to a file whose name is taken from the program arguments. For this we use a few utility functions and our `main()` will always have the following structure:

```c
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(_hpdftbl_jmp_env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_<EXAMPLE NAME>(pdf_doc, pdf_page);
    if( -1 == stroke_to_file(pdf_doc, argc, argv) )
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}
```

In order to make the example code consistent and focused on the table library and not on the general creating of PDF document we will include the supporting Haru set-up code in an include file and instead of the `main()` function shown above we will replace it with a macro with one parameter; the table function to be called to set-up the table example (see TUTEX_MAIN()).

All our example code will therefore be a fully standalone programs but structured in way not to obscure the actual table creation with a lot of boiler-plate PDF set-up code. All tutorial example programs `tut_ex<nn>` will therefore have the following general structure:

```c
#include "unit_test.inc.h"

void
create_table_XXXX(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    ...
}
TUTEX_MAIN(create_table_XXXX, FALSE)
```

The second argument to the TUTEX_MAIN() macro determines if the example should be generated with gridlines on the paper. This is useful for precisely position the table on a page.

In the `examples` directory the full source code for the setup and stroke functions can be found in all the tutorial examples, for example tut_ex01.c. They are very basic and follows the standard hpdf library methodology. The `setup_hpdf()` creates a new document wth one A4 page and the `stroke_to_file()` strokes the document to an output file which depends on the program argument.

Note

> If any of the test programs are run without any arguments the output file will be stored in the `out` directory
> and have the same name as the basename of the program with a "∗.pdf" suffix. If exactly one filename is
> specified as an argument then this is the file the output will be written to.

In the following we will focus only on the `create_table_<NAME_OF_EXAMPLE>()` function which will use
the two parameters `pdf_doc` and `pdf_page` to refer to the document and page to construct the table.

Note

> In order to make the examples robust and compatible with both Windows and Linux/OSX systems some
> conditional compilation instructions are also used, but we will not display them while discussing the basic
> usage to keep the focus on what matters.

The full source for all example are available in the `examples/` directory as well as in the Examples section of this
manual.

3.2 Your first table

tut_ex01.c

The first example shows the absolute most basic usage. We create a 2x2 table in steps as follows. We will follow
the framework oulined above. Our first example is tut_ex01.c

First we construct a table handle for a 2x2 table
```
create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
```

Here we note that:

- The size of the table has to be determined before the table handle is created

- All table function will refer to this handle, and we will always use the variable name `tbl` for this handle

- We use `size_t` instead of `int` since the table dimension is a size and as such can never be negative. In C
  it is always good practice to use `size_t` for positive numeric entities.

Once we have the table handle we can start to add content in these cells. For now lets just put a string that indicates
the cells position.
```
    hpdftbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
    hpdftbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
    hpdftbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
    hpdftbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
```

Here we note that:

- Cells are referred to starting from the top left cell that is cell (0x0).

- The `NULL` argument (4th argument) will be explained shortly.

Now It's time to size and position the table on the page. As a minimum you must specify the `x` and `y` position as well
as the width of the table. The library is smart enough to automatically figure out the height (but it is also possible
to force a larger height than strictly necessary either by specifying an overall table height or a minimum row height
using hpdftbl_set_min_rowheight())

The native coordinate system for PDF pages are given as the printing unit of DPI or *dots per inch*. By default, the
resolution of a PDF is 72 DPI.

To make it easier to directly set the size and position in centimeters a convenience function `hpdftbl_cm2dpi()`
can be used.

Note

> For precision positioning it is more accurate to give the position and sizes in dots directly.

In this example we set the size and position in centimeters. The paper coordinate system has its origin in the lower left corner of the paper. We position the top left of the table *1 cm* below and *1 cm* to the right of the top left corner of the paper. To make this easier we make use of the constant `A4PAGE_HEIGHT_CM` and make the table *5 cm* wide as follows:

```
HPDF_REAL xpos = hpdftbl_cm2dpi(1);
HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
HPDF_REAL width = hpdftbl_cm2dpi(5);
HPDF_REAL height = 0;  // Calculate height automatically
```

Now, there are several important observations to be made here:

- The origin of the paper coordinate system is bottom left which is (0,0)

- The anchor position by default is the top-left corner of the table (this can be adjusted by calling `hpdftbl↵ _set_anchor_top_left(FALSE)` function which will make the bottom left the anchor point instead)

- We use a predefined constant `A4PAGE_HEIGHT_IN_CM` to position the table vertically 1 cm from the top of the paper

- We let the library calculate the minimum table height automatically (based on the font height used in the table)

Now the only thing remaining is to print or stroke the table to the page and use the macro to create a main function `TUTEX_MAIN()` as follows:

```
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex01, FALSE)
```

and we are done!

If we put it all together it will give us the following basic table creation code

```
1
4  #include "unit_test.inc.h"
5
9  void
10 create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
11     const size_t num_rows = 2;
12     const size_t num_cols = 2;
13
14     hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
15
16     hpdftbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
17     hpdftbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
18     hpdftbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
19     hpdftbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
20
21     // We have to specify the top left position on the PDF as well as the width.
22     // We let the library automatically determine the height of the table based
23     // on the font and number of rows.
24     HPDF_REAL xpos = hpdftbl_cm2dpi(1);
25     HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
26     HPDF_REAL width = hpdftbl_cm2dpi(5);
27     HPDF_REAL height = 0;  // Calculate height automatically
28
29     // Stroke the table to the page
30     hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
31 }
32
33 TUTEX_MAIN(create_table_ex01, FALSE)
```

The generated table is shown in **Figure 1.** ([tut_ex01.c](tut_ex01.c))

| Cell 0x0 | Cell 0x1 |
| Cell 1x0 | Cell 1x1 |

Figure 1: *Your first table.*

As we explained above the coordinate system is in postscript dots. For precision positioning it might be useful to visualize this grid on the page. By using the `hpdftbl_stroke_grid()` function such a grid can be displayed on a page to help with positioning.

In our infrastructure set-up this call is controlled by setting the secon macro parameter to `TRUE`, i.← e. `TUTEX_MAIN(create_table_ex01, FALSE)`

If we add the grid to the page and show the upper left area of the paper with the grid we can view its positioning in the grid as shown in **Figure 2.**

Figure 2: *Your first table in the page coordinate system showing the upper left part of the paper.*

Since this is an A4 page it will have a height of roughly 841 points or 29.7cm. In our setup it is possible to generate thegrid by setting the third argument to `setup_hpdf()` to `TRUE`. This can be done by updating the `TUTEX_MAIN()` macro

3.3 Your second table - disconnecting program structure from data

One drawback of the program in the first example above is that if we want to have a different table size we need to actually change the code since we need one function call to store the data to be displayed in each cell. Wouldn't it be better if we could just supply an array with the data we want to display?

The function to do just that is
`hpdftbl_set_content(hpdftbl_t tbl, char **content)`

The content data is a 1-dimensional array of string pointers. Where each row is consecutive in the array. For example to create dummy data indicating what array position goes into what cell you could use the following setup:

Note

> We allocate each string dynamically in the dummy-data and since the program is just an illustration and terminates after the page has been created we never bother to free this memory. In a real life scenario this would of course be crucial!

We could then augment example 01 using this more efficient way to specify data as so:

```
create_table_ex02(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, 2, 2);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

tut_ex02.c

Running the code above in our infrastructure will give

Figure 3: *Specifying data in a table with an array of string pointers.(tut_ex02.c)*

In the above (small) example it might not have been a big safe but if you have a table with 20x10 rows ∗ cols then you will soon appreciate this way of specifying data.

There is even one more way of specifying data that in some situations are more efficient and allows a clear division between the table structure and look&feel and its data. This more efficient way is achieved by using cell callbacks either directly in individual cells or in one go by specifying the entire table as a data structure by using the `hpdftbl_stroke_from_data()` function. This will be described later when we discuss how to use callback functions.

But now it is time to explain the `NULL` value in the first example when we specified the content with the `hpdftbl_set_cell()` function.

3.4   Adding a header row

While it is possible (as discussed in section Style and font setting and Fonts and Colors) to manually adjust the font, size, style, background etc. on each cell individually there is a convenient shortcut to create a basic table with a header using the `hpdftbl_use_header()` function. By modifying the code above and add this line we get the following code and resulting table

```
create_table_ex02_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    content_t content;
    setup_dummy_content_with_header(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

The resulting table can be seen in **Figure 4**. We also modified the dummy data to have the work "Header" text for `row==0` in the first row (for details see tut_ex02_1.c)

Figure 4: *Adding automatic header formatted row (tut_ex02_1.c)*

3.5   Using labels in the table cells

A variant of a table is to present data with a short label describing what kind of data is displayed. This is often used when a table is used to present a data form. An example of this is shown in **Figure 4.** below.

Figure 4: *Specifying labels for each cell. (tut_ex03.c)*

Adding labels requires three things:

1. Enable the "label" feature with a call to `hpdftbl_use_labels(tbl, TRUE);`

2. Add the text that should be the label. Specifying these labels can either be done using the `hpdftbl_set_cell()` function as in
```
hpdftbl_set_cell(tbl, 0, 0, "Label 1", "Cell 0x0");
hpdftbl_set_cell(tbl, 0, 1, "Label 2", "Cell 0x1");
hpdftbl_set_cell(tbl, 1, 0, "Label 3", "Cell 1x0");
hpdftbl_set_cell(tbl, 1, 1, "Label 4", "Cell 1x1");
```
   or it can be done using the analog of specifying the labels in an array using the function `hpdftbl_set_labels()`.

3. In addition, there is one more key setting and that is whether the left cell border should be the whole cell or just the table height as was shown in **Figure 4.** above. This option is specified with `hpdftbl_use_labelgrid()`.

4. By default, the left border is from top to bottom. The differences between the two variants is shown in **Figure 5.** below.

Figure 5: *The two variants of left cell border with labels.*

Note

> Except for the simplest of tables both the table content and the labels should be specified in an array.

To create dummy date for both content and labels we use the function setup_dummy_content_label()

```
void setup_dummy_content_label(content_t *content, content_t *labels, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    *labels = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            snprintf(buff, sizeof(buff), "Label %zu:", cnt);
            (*labels)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
```

In the same way as before we call the functions to specify both the content and the labels (strictly speaking the call to hpdftbl_use_labelgrid() is not necessary since by default the short gridlines will be enabled when labels are first enabled.)

```
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);
```

and finally we also enable labels and the short variant of the left cell border

```
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
```

the remaining code we can leave untouched. With this we get the result shown in **Figure 4.** with the full code for the table shown below.

```
create_table_ex04(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);

    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

tut_ex04.c

3.6   Adding a table title

We have one last part of the table we haven't yet used and that is the table title. In the previous examples we created a table using hpdftbl_create() but there is also hpdftbl_create_title(). A title can also be added to an existing table (or perhaps updated) using hpdftbl_set_title()

To create a table with a title

```
    char *table_title = "tut_ex05: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
```

A table title occupies the top of the table in its own row which isn't part of the counting if the normal columns.

Figure 6: *Adding a title for the table. (tut_ex05.c)*

It is possible to adjust the colors, font-properties, and alignments of the title with two additional functions hpdftbl_set_title_style() and hpdftbl_set_title_halign()

The complete code for this example is shown below
```
create_table_ex05(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex05: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

3.7 Adjusting fonts and colors

The one thing we have skipped over so far and just used the defaults is the look & feel of the table as far as colors and fonts go. It is possible to adjust these setting at several levels of granularity. It is possible to:

1. Adjust the entire table in one go using hpdftbl_set_content_style()

2. Adjust one entire column using hpdftbl_set_col_content_style()

3. Adjust one entire row in using hpdftbl_set_row_content_style()

4. Adjust individual cells using hpdftbl_set_content_style()

It is also possible to adjust the color and thickness of the borders, but we will not discuss this more here and instead refer the reader to the API documentation.

Note

> We should also mention that there is a concept of a look & feel theme for the table which can be used to adjust all the parameters at once. This is discussed in Using themes.

Chapter 4

Cell and row spanning

The table can be modified both by adjusting the width of columns and how many rows and columns a cell is spanning.

4.1 Cell and row spanning

A common way to modify a table is to have a cell spanning either multiple columns, multiple rows or both. This is done using the function

```
int
hpdftbl_set_cellspan(const hpdftbl_t tbl,
                     size_t r, size_t c,
                     size_t rowspan, size_t colspan)
```

The specified `(r,c)` is the row and column of the upper left cell in merged cell that spans `rowspan` rows and `colspans` columns. This is also the row and column coordinates used to accessing the combined cell.

To illustrate this we will create a table with seven rows and five columns. We will merge three cells using cell-spanning as follows:

```
hpdftbl_set_cellspan(tbl, 0, 0, 1, 3);
hpdftbl_set_cellspan(tbl, 2, 2, 3, 3);
hpdftbl_set_cellspan(tbl, 3, 0, 4, 1);
```

For the data we will use the same setup as in tut_ex06.c This will then give the result shown in **Figure 8.**

Figure 8: ∗Having cells spanning multiple rows and columns. tut_ex07.c∗

4.2 Adjusting column width

By default, or column widths are divided equally regardless of the content. The width can be adjusted by explicitly set the relative width of a column as a percentage of the total table width. This is done with the function

```
int
hpdftbl_set_colwidth_percent(const hpdftbl_t tbl,
                             const size_t c,
                             const float w);
```

The width is set as a percentage of the total width and is specified as a floating point value in the range [0.0, 100.0]. An example of this is shown in **Figure 9.** below. An arbitrary number of columns can be given a width. For best result leave at least one column undefined and whatever remains of the table width will be assigned to that column. There is an error to try to specify a total column width > 100%.
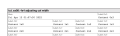
Figure 9: ∗Adjusting width of first columns. tut_ex08.c ∗

Chapter 5

Using callbacks

In the "[Getting started](GettingStarted.md)" chapter we discussed the preferred way to specify data and labels in table using data arrays. This is a very good way to populate a table in the cases the data is fairly static.

For data that is more dynamic and determined at runtime it is of course possible to construct the data array but the table library have one better way to do this and that is to set up label and content callbacks.

5.1   Introducing content callback functions

Content callbacks are functions that are called by the library for each cell and returns a string which is used as the data to be displayed. The signature for a cell content callback is defined by the type `hpdftbl_content_↩ callback_t` which is a pointer to a function defined as:

```
typedef
char * (*hpdftbl_content_callback_t)(void *, size_t, size_t);
```

This signature is also used for label callbacks. For style setting callback the signature is instead defined as

```
typedef
_Bool (*hpdftbl_content_style_callback_t)(void *, size_t, size_t, char *content, hpdf_text_style_t *);
```

To understand this lets start defining a callback function to specify content (or a label) that follows this signature.

```
char *
my_cell_cb(void *tag, size_t row, size_t col) { ... }
```

The parameters in the callback are

Parameter	Description
tag	Since a callback sometimes must know from what table or in what circumstances it is called it is possible to add a "tag" to ech table. This could be something as simple as pointer to a numeric identifier that uniquely identifies the table or perhaps a pointer to some function that retrieves data for this particular table.
row	The cell row
col	The cell column

It is possible to specify a callback to adjust content, style, and labels. A callback function can be specified to be used for every cell in the table or only for a specific cell. This can also be mixed in order to have, for example, one generic callback for most cells and have a different callback for a specific cell. Any callback set for a cell will override the callback set for the table

The API to specify these callbacks are:

API	Description
hpdftbl_set_content_cb()	Specify a content callback for the entire table.
hpdftbl_set_content_style_cb()	Specify a style callback for the entire table.
hpdftbl_set_label_cb()	Specify a label callback for the entire table.
hpdftbl_set_cell_content_cb()	Specify callback for an individual cell. A cell callback will override a potential table callback.
hpdftbl_set_cell_content_style_cb()	Specify a style callback for an individual cell. A cell callback will override a potential table callback.
hpdftbl_set_canvas_cb()	This is an advanced callback to allow for low level painting directly on the canvas that is the cell area arguments to the callback is different as it includes the bounding-box for th cell area. We will not further discuss this.

Note

> **Returned content string**. The string pointer returned from a callback is never stored in the table. only printed. It is therefore perfectly possible to have a static allocated buffer in the callback function that is used to construct the content and returned from the callback.

5.2 A content callback example

Let's now construct a simple example where the content and the labels are specified with callbacks.

We will create callbacks that will add a date string to the top left cell and just som dummy content in the rest of the cells. We could do this in two ways.

1. Add a generic table callback for all cells and then in that callback check if the row and column is (0,0) i.e. top-left and in that case create a date.

2. Add a generic table callback for all cells and then add a specific cell callback with the date for the (0,0) cell.

To illustrate both methods we will use method 1 for the labels and method 2 for the content.

Let's first create the date callback functions we need to add a date in the top left corner of the table that reflects the current date and time.

```
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    time_t t = time(NULL);
    ctime_r(&t, buf);
    return buf;
}
```

This would be sufficient for normal usage. However, the source code in tut_ex06.c that illustrates callbacks is slightly different:

```
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
```

```
        }
}
```

The reason for this is that all these examples also serve as unit tests for the library. The way the unit tests work is by comparing the output from all these examples with stored, manually checked "correct" versions of the output. Since any date changes will make the file different we must make the dates a known value whe the examples are run as unit teets. This we know when the flag `run_as_unit_test` is true and in that case a "dummy" static date is used.

The content and label functions can then be written as follows

```c
static char * cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    snprintf(buf, sizeof buf, "Content %02zu x %02zu", r, c);
    return buf;
}
static char * cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) { // Top-left cell
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
    return buf;
}
```

We note that we ignore the tag argument. Since we only have one table there is no need to use a tag to different from which table the callback comes.

For the table structure we will re-use our previous example and create a 2x2 table, and we get the following table creation code:

```c
create_table_ex06(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex06: 2x2 table with callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

Running this example gives the result shown in **Figure 7.** below, the full source code can be found in tut_ex06.c

Figure 7: *Using callbacks to populate the table and labels.*

5.3   Dynamic (late binding) callbacks

Warning

> This is an advanced concept and while simple in theory it does have some hidden "gotchas".

All callback functions discussed above must exist att compile time so that the address of the functions can be determined by the compiler. As we will discuss later it is possible to define a table as a data structure to avoid having to write several lines of code in defining a table.

Such a data structure could in theory be stored in a database or as a text file. In that case it will not be possible to specify a callback function since the address of function is determined at link time.

Fortunately it is possible to specify a function name (as a string) and have the standard C-library locate where that function is stored and return a pointer to it. This pointer is then the same as if the callback had been bound at compile time.

There is a analog set of functions that takes a string name of the function and looks up the actual function pointer and set that as the callback.

Those analogue functions are

API	Description
hpdftbl_set_dlhandle()	Option to set dynamic lib handle
hpdftbl_set_content_dyncb()	Table content late binding
hpdftbl_set_label_dyncb()	Table label late binding
hpdftbl_set_cell_label_dyncb()	Table cell label latex binding
hpdftbl_set_content_style_dyncb()	Table style late binding
hpdftbl_set_cell_content_style_dyncb()	Table cell content late binding
hpdftbl_set_cell_canvas_dyncb()	Cell canvas callback

They are identical to hhe already described "ordinary" setting callback functions with the difference these functions take a string as argument rather than a function pointer.

5.3.1 Using late binding

Using late binding is very similar to what we did in the previous examples, We start by defining the callback functions we need

```c
char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    snprintf(buf, sizeof buf, "Content %02zu x %02zu", r, c);
    return buf;
}
char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date created:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
    return buf;
```

```
}
```

It is however one crucial detail that cannot be overlooked. **None of the callbacks functions can be static!** If they are static they won't be found

Then it is really simple. We create the table with the function that should now be familiar and then add the callbacks with the names of the callback functions as so

```
hpdftbl_set_content_dyncb(tbl, "cb_content");
hpdftbl_set_label_dyncb(tbl, "cb_labels");
hpdftbl_set_cell_content_dyncb(tbl, 0, 0, "cb_date");
```

In this way it is possible to specify the entire table structure as a text structure that could be stored in a database or as a text file with just the name of the callback functions. However, care must be taken that they are named exactly as they are specified.

The complete table function is shown below and the full example be found in tut_ex30.c

```
void
create_table_ex30(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex30: Table with dynamic callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_dyncb(tbl, "cb_content");
    hpdftbl_set_label_dyncb(tbl, "cb_labels");
    hpdftbl_set_cell_content_dyncb(tbl, 0, 0, "cb_date");
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

Chapter 6

Error handling

All library function will return an error code < 0 and also set a global variable to a specific error code that can later be read by an error handler. In order to translate the error to a human-readable string the function `hpdftbl_get_last_errcode()` can be used as the following error handling snippet exemplified by a call to `hpdftbl_set_colwidth_percent()`

```c
if( hpdftbl_set_colwidth_percent(tbl, 5, 110) ) {
    // This is an error
    char *err_str;
    int err_code, r, c;
    err_code=hpdftbl_get_last_errcode(&err_str, &r, &c);
    if( err_code ) {
        printf("*ERROR*: \"%s\" at cell (%d, %d)",err_str,r,c);
        exit(1);
    }
}
```

As can be seen from the snippet above it would yield quite long winding error handling if one where to check every single library call. Instead, there is the option of installing an error handler that would be called in the event of an error.

The table error handle has the signature

```c
void hpdftbl_error_handler_t)(hpdftbl_t tbl, int r, int c, int err)
```

Where the arguments are

Argument	Description
tbl	The table in where the error happened. **Note:** This might be `NULL` since not all errors happen within the context of a table
r, c	The row and column if the error happens in a specified cell, otherwise these will be (-1,-1)
err	The internal error code. This si always a negative number.

The error handler is set with the `hpdftbl_set_errhandler()` method. An example of a very simple error handle is:

```c
void
my_table_error_handler(hpdftbl_t t, int r, int c, int err) {
    if( r>-1 && c>-1 ) {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)\n", err, hpdftbl_get_errstr(err), r,
        c);
    } else {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" \n", err, hpdftbl_get_errstr(err));
    }
    exit(EXIT_FAILURE);
}
```

In the above error handler we have made use of the utility function `hpdftbl_get_errstr()` that translates the internal error code to a human-readable string.

In fact this exact error handler is available as a convenience in the library under the name `hpdftbl_default↩_table_error_handler` so to use this trivial error handler just add the following line to your code

```
hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
```

More advanced error handler must be written for the particular application they are to be used in.

6.1 Using emulated exception handling

As can be ssen above the default error handler terminates the running process with a call to `exit(EXIT_↩FAILURE)`. Terminating te process might not always be approriate (especially for a daemon process). An alternative way to handle a fault state is to use `setjmp()`/`longjmp()` to simulate an exception handling.

In the program setup code a jump point is established and then if an error is detected the error handler will jump to the set jump point.

For example, all tutorial examples share the same `main()` function as shown below

```
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    run_as_unit_test = 2==argc ;
    if (setjmp(_hpdftbl_jmp_env)) {
        return EXIT_FAILURE;
    }
    hpdftbl_set_errhandler(table_error_handler);
    setup_hpdf(&pdf_doc, &pdf_page, _showgrid_);
    _tbl_(pdf_doc, pdf_page);
    if( -1 == stroke_to_file(pdf_doc, argc, argv) )
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}
```

The relevant part here is the `setjmp(_hpdftbl_jmp_env)` code which establish a jump destination. An basic error handler that uses this could now look like this:

```
static void
table_error_handler(hpdftbl_t t, int r, int c, int err) {
    if (r > -1 && c > -1) {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)\n", err, hpdftbl_get_errstr(err), r,
        c);
    } else {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" \n", err, hpdftbl_get_errstr(err));
    }
    longjmp(_hpdftbl_jmp_env, 1);
}
```

If an error occur the `longjmp()` will come to the `setjmp()` point and since it returns the value of `1` it will enter the if-statement free the doc structure and then terminate the process by exiting the `main()` function.

In a more complex program it might be useful to instead of exiting the process give the user an error message, do any cleanup (such as freeing the PDF document) and try again if this perhaps was a recoverable error.

The actual error handler used in the tutorial examples is slightly longer as it prints all available information from the error handling "subsystem" such as which file and line number (in the library) where the error was triggered and any optional extra information was given in regard to the error mode. In addition, a stacktrace is also generated to `stderr`

The real (production grade) error handler therefore looks as shown below

```
table_error_handler(hpdftbl_t t, int r, int c, int err) {
    int lineno;
    char *filename;
    char *extrainfo;
    hpdftbl_get_last_err_file(&lineno, &filename, &extrainfo);
    if (r > -1 && c > -1) {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)", err,
                hpdftbl_get_errstr(err), r, c);
    } else {
        fprintf(stderr, "*** Table Error: [%d] \"%s\"", err,
                hpdftbl_get_errstr(err));
    }
```

```c
    if( filename != NULL ) {
        fprintf(stderr," in %s:%d",filename, lineno);
    }
    if( extrainfo != NULL ) {
        fprintf(stderr,". Info: \"%s\"\n",extrainfo);
    }
    else {
        fprintf(stderr,"\n");
    }
    // Also print the available stacktrace
    void* callstack[128];
    int i, frames = backtrace(callstack, 128);
    char** callstack_sym = backtrace_symbols(callstack, frames);
    if( callstack_sym != NULL ) {
        fprintf(stderr, "Stacktrace:\n");
        for (i = 0; i < frames; ++i) {
            fprintf(stderr, "%s\n", callstack_sym[i]);
        }
        free(callstack_sym);
    }
    longjmp(_hpdftbl_jmp_env, 1);
}
```

Note

> A common way to extend the error handling is to log the errors to syslog. When the library is used on OSX from 11.0 and onwards it should be remembered that OSX is broken by design as far as syslog logging is concerned. Apple in its wisdom introduced "Unified logging" which breaks the `syslog()` function and no logging is ever produced in the filesystem directly (i.e. to `/var/log/system.log`).
> Instead, the only way to view the logs is by using the utility `log`. So in order to view the log from a particular application the following command has to be given
> ```
> log stream --info --debug --predicate 'sender == "APPLICATION_NAME" --style
> syslog
> ```

6.2 Additional information

When an error is triggered the file name and line number in the library where the error was triggered is saved as well as an optional information string that some error states might set.

All this extra information can be retrieved by the library function hpdftbl_get_last_err_file()

Note

> The file name and line number displayed is always the point in the library that discovered the error state. It does not indicate the file name and line number of the client code that triggered the error as the error is discovered in the library routines.

6.3 Translating HPDF error codes

The standard error handler for the plain HPDF library is specified when a new document is created, for example as'
```c
...
pdf_doc = HPDF_New(error_handler, NULL);
HPDF_SetCompressionMode(pdf_doc, HPDF_COMP_ALL);
...
```

The error handler signature is defined by Haru PDF library as
```c
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data);
```

It is then up to the application code to decide how to handle the error. To simplify the handling of core HPDF error the library also offer a convenience function to translate the Haru library error code into a human-readable string. This function is
```c
const char *
hpdftbl_hpdf_get_errstr(const HPDF_STATUS err_code)
```

and is used in the error handler in all the examples.

6.4 Example of setting up error handler

The following table creation code have a deliberate error in that it tries to assign a total column width of more than 100% which of course isn't possible.

```
void
create_table_ex10(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_colwidth_percent(tbl, 0, 30);
    hpdftbl_set_colwidth_percent(tbl, 1, 30);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

This is available in the example directory as tut_ex10.c.

If we simulate a "typo" and add a deliberate error by making the column widths larger than 100% by writing

```
hpdftbl_set_colwidth_percent(tbl, 0, 80);
hpdftbl_set_colwidth_percent(tbl, 1, 30);
```

When this code is then executed the following will be printed to standard error and the process will be stopped.

```
*** Table Error: [-12] "Total column width exceeds 100%"
```

Chapter 7

Font and style setting

The format of each cell can be adjusted with respect to:

1. Font-family and style (size, bold, italic etc.)

2. Font- and background-color

3. Border thickness and color

In this section we will focus on how to adjust the font and background color. The style can be adjusted both for the entire table at once and also for individual cells. The individual cell style will always override the table cell style.

The primary API to adjust the table style are:

```
// Set background color for entire table
int hpdftbl_set_background(hpdftbl_t t,
                           HPDF_RGBColor background);
// Set label style for the entire table
int hpdftbl_set_label_style(hpdftbl_t t,
                            char *font, HPDF_REAL fsize,
                            HPDF_RGBColor color, HPDF_RGBColor background);
// Set content style for entire table
int hpdftbl_set_content_style(hpdftbl_t t,
                              char *font, HPDF_REAL fsize,
                              HPDF_RGBColor color, HPDF_RGBColor background);
// Set content style for specified cell
int hpdftbl_set_cell_content_style(hpdftbl_t t,
                                   size_t r, size_t c,
                                   char *font, HPDF_REAL fsize,
                                   HPDF_RGBColor color, HPDF_RGBColor background);
// Set content style for specified row in table
int hpdftbl_set_row_content_style(hpdftbl_t t,
                                  size_t r,
                                  char *font,  HPDF_REAL fsize,
                                  HPDF_RGBColor color, HPDF_RGBColor background);
// Set content style for specified column in table
int hpdftbl_set_col_content_style(hpdftbl_t t,
                                  size_t c,
                                  char *font,  HPDF_REAL fsize,
                                  HPDF_RGBColor color, HPDF_RGBColor background);
```

7.1   Adjusting fonts and colors

Fonts are specified as a string with the type font family name as recognized by the core Haru PDF library, e.g. "Times-Roman", "Times-Italic", "Times-Bold" etc. As a convenience not to have to remember the exact font name strings the following three font family are defined as $HPDF\_FF\_*$ where the last part of the name is specified as the following table shows

Font family	Italic	Bold	BoldItalic
TIMES	TIMES_ITALIC	TIMES_BOLD	TIMES_BOLDITALIC
HELVETICA	HELVETICA_ITALIC	HELVETICA_BOLD	HELVETICA_BOLDITALIC
COURIER	COURIER_ITALIC	COURIER_BOLD	COURIER_BOLDITALIC

Table 1: *Predefined font family and variants*

So to use the "Helvetic" font family the constant `HPDF_FF_HELVETICA` is used and so on.

Colors are specified in the standard Haru way, i.e. as an instance of the structure `HPDF_RGBColor`. As another convenience the following colors are predefined

```
#define HPDF_COLOR_DARK_RED       (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
#define HPDF_COLOR_RED            (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
#define HPDF_COLOR_LIGHT_GREEN    (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
#define HPDF_COLOR_GREEN          (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
#define HPDF_COLOR_DARK_GREEN     (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
#define HPDF_COLOR_DARK_GRAY      (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
#define HPDF_COLOR_LIGHT_GRAY     (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
#define HPDF_COLOR_XLIGHT_GRAY    (HPDF_RGBColor) { 0.95f, 0.95f, 0.95f }
#define HPDF_COLOR_GRAY           (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
#define HPDF_COLOR_SILVER         (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
#define HPDF_COLOR_LIGHT_BLUE     (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
#define HPDF_COLOR_BLUE           (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
#define HPDF_COLOR_DARK_BLUE      (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
#define HPDF_COLOR_WHITE          (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
#define HPDF_COLOR_BLACK          (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
```

So for example to set the overall default font to 12pt Times Roman with black text on white bottom the following call must be made

```
...
hpdftbl_set_content_style(tbl, HPDF_FF_TIMES, 12, HPDF_COLOR_BLACK, HPDF_COLOR_WHITE);
...
```

Since RGB for colors are specified as a floating point number in range [0.0, 1.0] and most color tables give colors as an integer triple there is exists a macro to make this conversion easier

```
#define HPDF_RGB_CONVERT(r,g,b) (HPDF_RGBColor){r/255.0,g/255.0,b/255.0}
```

which will allow the easier specification of color such as

```
#define HPDF_COLOR_ORANGE           HPDF_RGB_CONVERT(0xF5, 0xD0, 0x98);
#define HPDF_COLOR_ALMOST_BLACK     HPDF_RGB_CONVERT(0x14, 0x14, 0x14);
```

7.2 Using style callbacks

In much the same way as callbacks can be used for specifying content and labels so can a callback be used to specify the style of a cell or the entire table.

A style callback has the following signature

```
_Bool
hpdftbl_content_style_callback_t(void *tag, size_t r, size_t c, char *content, hpdf_text_style_t *style);
```

In order for the settings to be applied the callback has to return a boolean `TRUE` value.

If the callback returns `FALSE` the settings will **not** be applied.

The parameters are used as follows:

- The `tag` parameter has the same meaning as for content and label callbacks; an optional unique identifier for the table.** The `tag` parameter should always be checked for possible `NULL` value since it is not required for a table to have a tag.

- The `r` and `c` arguments are the row and column of the cell the callback is made for

- The `content` is the cell content string. The rationale for including this in the style callback is to allow for highlighting in the table of specific data. It could for example be something as simple as wanting to mark all values above a certain threshold with another background color in the table to draw attention.

- Finally, the actual style is encompassed by the `hpdf_text_style_t` and is defined as the following structure

```c
typedef struct text_style {
    char *font;
    HPDF_REAL fsize;
    HPDF_RGBColor color;
    HPDF_RGBColor background;
    hpdftbl_text_align_t halign;
} hpdf_text_style_t;
```

The style callbacks can exactly as the content callback be specified for either the entire table or for a specific cell. A cell callback will always override a table callback. The two functions to set up style callbacks are

```c
int
hpdftbl_set_cell_content_style_cb(hpdftbl_t tbl,
                                  size_t r, size_t c,
                                  hpdftbl_content_style_callback_t cb);
int
hpdftbl_set_content_style_cb(hpdftbl_t tbl,
                             hpdftbl_content_style_callback_t cb);
```

Note

> Due to som technicalities **the style callbacks are called twice** per cell. The first call is necessary to set up the background canvas and at that stage the content is not necessarily known since it could be later specified with a content callback. The first time the callback is made the `content` parameter is always guaranteed to be `NULL`

7.2.1   Style callback example

An example of a callback function to set a background color for a header row/column for a table could for example be done as follows

```c
_Bool
cb_style(void *tag, size_t r, size_t c, char *content, hpdf_text_style_t *style) {
    // Format the header row/column with a grey background and Helvetica font while the rest of the
    // table uses "Times Roman"
    if( 0==r || 0==c ) { // Headers
        style->font = HPDF_FF_HELVETICA_BOLD;
        style->fsize = 12;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_LIGHT_GRAY;
        if ( c > 0 )
            style->halign = CENTER;
        else
            style->halign = LEFT;
    } else { // Content
        style->font = HPDF_FF_TIMES;
        style->fsize = 11;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_WHITE;
        style->halign = CENTER;
    }
    return TRUE;
}
```

and the table setup code can then be written as

```c
void
create_table_ex09(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_content_style_cb(tbl, cb_style);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
```

```
}
```

The resulting table is shown in **Figure 10.** below.

Figure 10: *Using a style callback to highlight header rows & columns. tut_ex09.c*

7.3 Adjusting grid line styles

There are four distinct set of grid lines as far as the library is concerned.

1. The outer gridlines (or border) around the table, and

2. The inner vertical grid line

3. The inner horizontal grid line

4. The inner top grid line (not the outer border!)

All these types of gridlines are styled in the same way using the functions
```
int
hpdftbl_set_inner_tgrid_style(hpdftbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color,hpdftbl_line_dashstyle_t dashstyle);
int
hpdftbl_set_inner_vgrid_style(hpdftbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);
int
hpdftbl_set_inner_hgrid_style(hpdftbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);
int
hpdftbl_set_inner_grid_style(hpdftbl_t t,
                             HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);
```

Each type of gridlines can be adjusted with line width, color and style. The last function in the list, hpdftbl_set_inner_grid_style(), is a convenience function that sets both the vertical and horizontal inner lines in one call.

The table below illustrates the various dashed line styles available and their names. See also hpdftbl_dashstyle and grid style functions hpdftbl_set_inner_grid_style(), hpdftbl_set_inner_vgrid_style() and hpdftbl_set_inner_hgrid_style()

Dash Style	Illustration
LINE_SOLID	
LINE_DOT1	
LINE_DOT2	
LINE_DOT3	
LINE_DOT4	
LINE_DASH1	
LINE_DASH2	
LINE_DASH3	
LINE_DASH4	
LINE_DASH5	
LINE_DASHDOT1	
LINE_DASHDOT2	

The following example (tut_ex20.c) makes use of these settings as shown below
```
void
```

```
create_table_ex20(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 5;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_inner_vgrid_style(tbl, 0.7, HPDF_COLOR_DARK_GRAY, LINE_SOLID);
    hpdftbl_set_inner_hgrid_style(tbl, 0.8, HPDF_COLOR_GRAY, LINE_DOT1);
    hpdftbl_set_inner_tgrid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
    hpdftbl_set_outer_grid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(10);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

and when run will result in the following table:

Content 0	Content 1	Content 2	Content 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15
Content 16	Content 17	Content 18	Content 19

7.4 Adding zebra lines in a table

A common way to make it easier to read a table is to make every other row a different color. This is sometimes known as zebra lines (or rows). This can be easily accomplished in the library by using the functions

```
int
hpdftbl_set_zebra(hpdftbl_t t, _Bool use, int phase);
int
hpdftbl_set_zebra_color(hpdftbl_t t, HPDF_RGBColor z1,  HPDF_RGBColor z2);
```

The first function is used to enable/disable row coloring and the second to set the first and second color. The `phase` parameter determines if color 1 is used first or is color 2 is used on the first row. Setting phase tom0 will make the first row use color 1 as background.

The default color are white and light gray. The following example (tut_ex15.c) shows how this can be done:

```
void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_zebra(tbl, TRUE, 1);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

Running this example will give the following result

tut_ex15.c

Content 0	Content 1	Content 2	Content 3	Content 4
Content 5	Content 6	Content 7	Content 8	Content 9
Content 10	Content 11	Content 12	Content 13	Content 14
Content 15	Content 16	Content 17	Content 18	Content 19
Content 20	Content 21	Content 22	Content 23	Content 24
Content 25	Content 26	Content 27	Content 28	Content 29
Content 30	Content 31	Content 32	Content 33	Content 34

We can make a small modification by setting $\texttt{phase} = 1$ (instead of the default 0) to start with color2. In addition, we can adjust the inner horizontal gridlines to have the same extra light gray as the zebra line making them "invisible" by modifying the table setup as follows (tut_ex15_1.c).

```
void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    //hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_zebra(tbl, TRUE, 1);
    // Normal inner line (same color as default Zebra to make them "invisible"
    hpdftbl_set_inner_hgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY,LINE_SOLID );
    // Top inner line. Comment this line to get a visible top line
    hpdftbl_set_inner_tgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY,LINE_SOLID );
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

Running this gives the following result:

tut_ex15_1.c

Content 0	Content 1	Content 2	Content 3	Content 4
Content 5	Content 6	Content 7	Content 8	Content 9
Content 10	Content 11	Content 12	Content 13	Content 14
Content 15	Content 16	Content 17	Content 18	Content 19
Content 20	Content 21	Content 22	Content 23	Content 24
Content 25	Content 26	Content 27	Content 28	Content 29
Content 30	Content 31	Content 32	Content 33	Content 34

Note

Another way to hide a gridline is to set its width to 0.

Chapter 8

Using themes

A theme (or style theme) is a definition of the "look & feel" of a table. It doesn't affect the structure of the table such as the size of the table or how many columns or rows a cell spans. It is practical shortcut when many tables should be displayed in the same style. It allows the compact specification of the table by applying a theme to the table instead of having to call multiple functions to achieve the same thing. In addition, if the design should be changed there is only one place to update instead of for each table.

A theme can also be serialized to and from a file/string buffer

A theme controls the following aspects of a table

- The content and label text style

- The header and title text style

- The inner and outer border style

- The usage (or not) of labels and whether the shorter label grind lines should be used

- If a header row should be used or not

- If a title should be used or not

if you have multiple table in a document it is possible to create a *table theme* which consists of some core styling of a table that can be reused.

Note

> By design any specific settings for individual cells is not saved in a theme as a theme can be applied to any table regardless of the specific table structure.

All information for a theme is encapsulated in the hpdftbl_theme structure.

This structure can be set up manually and then applied to a table. However, the recommended way is to first use the "theme getter" function to get the default theme and then modify this default theme as needed since it allows you to only have to update the parts affected by a change.

The functions to work with a theme are :

API	Description
hpdftbl_apply_theme()	Apply the given theme to a table
hpdftbl_get_default_theme()	Get the default theme into a new allocated structure
hpdftbl_destroy_theme()	Free the memory used by a theme
hpdftbl_get_theme()	Extract a theme from specified table
hpdftbl_theme_dump()	Serialize a theme to a file
hpdftbl_theme_dumps()	Serialize a theme to a string buffert
hpdftbl_theme_load()	Load a serialized theme from file
hpdftbl_theme_loads()	Load a serialized theme from a string buffert

It is the responsibility of the user of the library to destroy the theme structure by ensuring that `hpdftbl_destroy_theme()` is called when a theme goes out of scope.

The default font styles for the default theme are shown in table 1.

Style	Font	Size	Color	Background	Alignment
content	HPDF_FF_COURIER	10	Black	White	Left
label	HPDF_FF_TIMES_ITALIC	9	Dark gray	White	Left
header	HPDF_FF_HELVETICA_BOLD	10	Black	Light gray	Center
title	HPDF_FF_HELVETICA_BOLD	11	Black	Light gray	Left

Table 1: *Default font styles.*

Theme parameter	Default value
use_labels	FALSE
use_label_grid_style	FALSE
use_header_row	FALSE
use_zebra	FALSE

Table 2: *Default table structure parameters.*

Border	Color	Width (pt)
inner_border	Grey	0.7
outer_grid	Dark Grey	1.0

Table 3: *Default border parameters.*

8.1   Example of serializing theme and table

In tut_ex41.c an example ofmhow to read a theme and table back from their serialized representation can be found as also shown below.

```
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
hpdftbl_theme_t theme;
if(0 == hpdftbl_load(tbl, "tests/tut_ex41.json")  ) {
```

```
    if(0 == hpdftbl_theme_load(&theme, "tests/tut41_theme.json")) {
        hpdftbl_apply_theme(tbl, &theme);
        hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
    } else {
        fprintf(stderr,"%s\n","Failed to load 'tests/default_theme.json'\n");
        exit(1);
    }
} else {
    fprintf(stderr,"%s\n","Failed to load 'tests/tut_ex41.json'\n");
    exit(1);
}
```

Chapter 9

Tables layout from data

So far we have constructed the layout of table by issuing API calls per table to set up, for example, the column widths and what cells should merge with what other cells and so on. Previously we saw that data to be put in the table could be specified by either directly issuing API calls per cell, using a 2D array that we populate with data and then finally use callbacks to generate the data in the cells.

The final and most powerful way of constructing a table is to define the table structure as data. This *structural data* together with a style theme can completely define a table.

This will allow the dynamic construction of tables with only one API call instead of the multiple call required to construct a table the usual way. It can initially seem more complex but for advanced table this is indeed a much simpler and easy to maintain. In fact, this will allow a table to be (almost, we'll get back to the limitations) defined entirely in a database and makes it possible to adjust tha table as the data changes without ever updating the code (or recompile).

9.1 Defining a table in data

There are two data structure that are used when defining a table. First there is a data structure for the overall table specifics and then in that structure a structure to specify the layout of each cell. In addition, a theme needs to be defined (see section Themes). It is possible to omit the theme by specifying `NULL` in which case the default theme will be used.

To stroke a table from data the following API call is used
```
int
hpdftbl_stroke_from_data(HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t tbl_spec, hpdftbl_theme_t
        *theme);
```

In order to populate the table with suitable data callback functions are used (see section Using callbacks)

The overall table is first defined as an instance of
```
typedef struct hpdftbl_spec {
    char *title;
    _Bool use_header;
    _Bool use_labels;
    _Bool use_labelgrid;
    size_t rows;
    size_t cols;
    HPDF_REAL xpos;
    HPDF_REAL ypos;
    HPDF_REAL width;
    HPDF_REAL height;
    hpdftbl_content_callback_t content_cb;
    hpdftbl_content_callback_t label_cb;
    hpdftbl_content_style_callback_t style_cb;
    hpdftbl_callback_t post_cb;
    hpdftbl_cell_spec_t *cell_spec;
```

```
} hpdftbl_spec_t;
```

Then each cell (referenced above in the `cell_spec` field) is defined as an instance of

```
typedef struct hpdftbl_cell_spec {
    size_t row;
    size_t col;
    unsigned rowspan;
    unsigned colspan;
    char *label;
    hpdftbl_content_callback_t content_cb;
    hpdftbl_content_callback_t label_cb;
    hpdftbl_content_style_callback_t style_cb;
    hpdftbl_canvas_callback_t canvas_cb;
} hpdftbl_cell_spec_t;
```

9.2 A first example of defining table as data

To understand how this is done lets start to define a basic 3x3 table with header row (so 4x3 in total) as data. First we create an instance of the table data

```
hpdftbl_spec_t tbl_spec = {
        // Title and header flag
        .title=NULL, .use_header=TRUE,
        // Label and labelgrid flags
        .use_labels=FALSE, .use_labelgrid=FALSE,
        // Row and columns
        .rows=4, .cols=3,
        // Position of the table, xpos and ypos
        .xpos=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
        // width and height
        .width=hpdftbl_cm2dpi(15), .height=0,
        // Content and label callback
        .content_cb=cb_content, .label_cb=cb_label,
        // Style and table post creation callback
        .style_cb=NULL, .post_cb=NULL,
        // Pointer to optional cell specifications
        .cell_spec=NULL
};
```

Note

> In the table definition we use the C99 feature of specifying the field name when defining data in a structure.

Then the actual API call is trivial compared to the table creation code we have seen in the previous examples and consists of only one line of code

```
void
create_table_ex13_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
}
```

The result is as expected and shown in **Figure 13** but with much less code!

***Figure 13:** ∗Defining a table with a data structure tut_ex13_1.c∗

9.3 A second example of defining a table as data

In the previous example we kept it simple didn't specify any format or content fór a table cell. Let us therefore create a slightly more complex example where we create a form which easily could be used to display data records from a DB.

The nice thing about separating layout and table structure from the data population in the callbacks is that this can almost be seen as a poor man's model-view-controller where the table structure is completely separate from the data (and how it is created).

A good way to start designing a table is to make a sketch on how it should look. Our goal is to crete the table structure as shown in the empty table in **Figure 14** below

Figure 14: Sketch of table to be designed

To get this layout we use a basic table with :

1. Five rows and four columns

2. No header and no title

3. We use labels and label grids

To make it easier to see how to construct the table we can overlay the sketch with a grid shown in blue in **Figure 15**. As can be seen this is a basic 5x4 table where a number of cells span multiple columns.

Figure 15: Sketch of table to be designed with 5x4 table overlaid

To start we set up the table specification as in the previous example with necessary changes. We will also need to specify cell specifications this time, and we assume those are available in an array of cell structures called `cell_specs`.

Before we specify the table structure we have one design decision to make. For the callbacks we can either use the table callback for all cells and check row and column to get the appropriate data, or we can add individual callbacks for each cell. The first case has the advantage to only need one callback function (but a lot of tests) and the second that each callback will be small and focused to get the data for that individual cell, but we will need potentially one callback for each cell unless there are commonalities between the cells so one callback can serve multiple cells. Remember that we still get the row and column as arguments in the callback so we weill always know exactly for which cell the callback was made.

To keep the size of this example we will use the table callback method for content and specify the label directly in the cell specification. With this decision made we get the following definition cell specifications

```
hpdftbl_cell_spec_t cell_specs[] = {
        {.row=0, .col=0, .rowspan=1, .colspan=3,
         .label="Name:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=0, .col=3, .rowspan=1, .colspan=1,
         .label="Date:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=1, .col=0, .rowspan=1, .colspan=4,
         .label="Address:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=2, .col=0, .rowspan=1, .colspan=3,
         .label="City:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=2, .col=3, .rowspan=1, .colspan=1,
         .label="Zip:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=3, .col=0, .rowspan=1, .colspan=4,
         .label="E-mail:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=4, .col=0, .rowspan=1, .colspan=2,
         .label="Work-phone:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=4, .col=2, .rowspan=1, .colspan=2,
         .label="Mobile:",
         .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        HPDFTBL_END_CELLSPECS // Sentinel to mark the end of
};
```

As can be seen we need to have an end of cell specification sentinel since we could decide to provide details for one or more cells and there is no way for the library to know how many fields to read otherwise. There is even a convenience constant in the library `PDFTBL_END_CELLSPECS` that can be used as the last record.

The overall table specification is pretty much as before but with the added cell specifications.

```
hpdftbl_spec_t tbl_spec = {
        // Title and header flag
        .title=NULL, .use_header=FALSE,
        // Label and labelgrid flags
        .use_labels=TRUE, .use_labelgrid=TRUE,
        // Row and columns
        .rows=5, .cols=4,
        // xpos and ypos
        .xpos=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
        // width and height
        .width=hpdftbl_cm2dpi(15), .height=0,
        // Content and label callback
        .content_cb=cb_content, .label_cb=cb_label,
        // Style and table post creation callback
        .style_cb=NULL, .post_cb=NULL,
        // Pointer to optional cell specifications
        .cell_spec=cell_specs
};
```

When this is run (see tut_ex13_2.c) it generates the following image, **Figure 16**

***Figure 16:** Specifying a table as data with cell specifications.*

What remains is to write the proper table content callback that will populate the table. In a real life scenario his data will most likely come from a database but adding that in our example would bring too far. Instead, we will just use some fake static dummy data to illustrate the principle.

Since we have one callback for all cells we need to test from which cell the call come from. Here is a very important point to make. **The row and column number will be the row and cell columns in the original table before any column or row spans was applied.** In this example it means that for example the "Date" field (upper right) will have `row=0` and `col=3` and **not** `(0,1)` !!.

With this information we can write the following (dummy) table callback

```
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char *cell_content[] =
            {"Mark Ericsen",
             "12 Sep 2021",
             "123 Downer Mews",
             "London",
             "NW2 HB3",
             "mark.p.ericsen@myfinemail.com",
             "+44734 354 184 56",
             "+44771 938 137 11"};
    if( 0==r && 0==c) return cell_content[0];
    else if (0==r && 3==c) return cell_content[1];
    else if (1==r && 0==c) return cell_content[2];
    else if (2==r && 0==c) return cell_content[3];
    else if (2==r && 3==c) return cell_content[4];
    else if (3==r && 0==c) return cell_content[5];
    else if (4==r && 0==c) return cell_content[6];
    else if (4==r && 2==c) return cell_content[7];
    else return NULL;
}
```

and we get the (expected) result as shown in **Figure 17** below.

***Figure 17:** Specifying a table as data with cell specifications and "dummy" data.*

The alternative of specifying individual callback for each cell would then require that each cell have a callback provided or perhaps even a mix with both a general table callback and selected cell callbacks.

The priority is such that a cell callback will always override a table callback. In the above example the callback for the name field could as an example be

```
static char *
cb_content_name(void *tag, size_t r, size_t c) {
    static char *cell_content = "Mark Ericsen";
    return cell_content;
}
```

Chapter 10

Widgets

10.1 Overview

A feature in the library is the possibility to add widgets in table cell. A widget is used to visualize da ata value in a cell instead of a numeric value. For example a percentage value can instead be represented by a horizontal bar.

As of this writing the library supports the following five widgets.

10.1.1 1. Segmented horizontal bar example

Horizontal discrete (segmented) bar. Number of segment is user defined.

■■■■□□□□□□ *40%*

10.1.2 2. Horizontal bar example

Basic horizontal bar

� *60%*
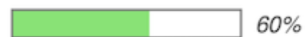
10.1.3 3. Signal strength meter example

A widget indicate a signal strength in similar fashion as the signal strength meter on a phone.

▂▃▅ signal

10.1.4   4. Radio sliding button example

Radio button/Slider with different on/off

10.1.5   5. Boxed letters example

Highlight zero or more letters

10.2   Widget functions

All the widgets are used in the same way. They are included as a part of a canvas callback function as installed by the hpdftbl_set_canvas_cb() and hpdftbl_set_cell_canvas_cb() functions. The callback function itself has to follow the canvas callback signature which is defined as

```
typedef void (*hpdftbl_canvas_callback_t)(HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_REAL, HPDF_REAL,
    HPDF_REAL,
                                  HPDF_REAL);
```

and a typical example of a canvas callback function, and it's installation would be

```
void
cb_draw_segment_hbar(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                  size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                  HPDF_REAL width, HPDF_REAL height)
{ ... }
...
hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_segment_hbar);
```

Each widget has its on function that should be included in the canvas callback to display and size the widget. The different widgets has slightly different defining functions depending on what they display and are defined as follows.

10.2.1   Segmented horizontal bar defining function

```
void
hpdftbl_widget_segment_hbar(const HPDF_Doc doc, const HPDF_Page page,
                         const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, const
    HPDF_REAL height,
                         const size_t num_segments, const HPDF_RGBColor on_color, const double
    val_percent,
                         const _Bool hide_val)
```

10.2.2   Horizontal bar defining function

```
void
hpdftbl_widget_hbar(const HPDF_Doc doc, const HPDF_Page page,
                 const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, const HPDF_REAL
    height,
                 const HPDF_RGBColor color, const float val, const _Bool hide_val)
```

10.2.3 Signal strength defining function

```
void
hpdftbl_widget_strength_meter(const HPDF_Doc doc, const HPDF_Page page,
                              const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, const
        HPDF_REAL height,
                              const size_t num_segments, const HPDF_RGBColor on_color, const size_t
        num_on_segments)
```

10.2.4 Radio sliding button defining function

```
void
hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
                            HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, _Bool state)
```

10.2.5 Boxed letters defining function

```
void
hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
                                    HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
                                    const HPDF_RGBColor on_color, const HPDF_RGBColor off_color,
                                    const HPDF_RGBColor on_background, const HPDF_RGBColor off_background,
                                    const HPDF_REAL fsize,
                                    const char *letters, _Bool *state)
```

10.3 Usage

The widget function is included in either a table canvas callback or more commonly in a cell canvas callback. Let's construct a basic example with a 1x2 table that shows a segmented horizontal bar indicating a fictive battery charge level and signal strength meter as shown in the figure below
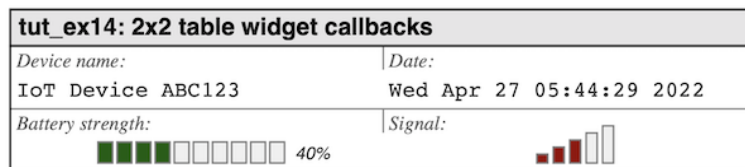
Figure 10.1 tut_ex14.c

For this we start by constructing the callback for the battery display. In a real application the value would probably be read from a database but here we just use a hard coded value

```
cb_draw_battery_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                       size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                       HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL segment_tot_width = width * 0.5;
    const HPDF_REAL segment_height = height / 3;
    const HPDF_REAL segment_xpos = xpos + 40;
    const HPDF_REAL segment_ypos = ypos + 4;
    const size_t num_segments = 10;
    const _Bool val_text_hide = FALSE; // Display the percentage
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_GREEN;

    // This should in reality be retrieved programmatically (for example from a DB)
    const double val_percent = 0.4;

    hpdftbl_widget_segment_hbar(
            doc, page, segment_xpos, segment_ypos, segment_tot_width,
            segment_height, num_segments, on_color, val_percent, val_text_hide);
}
```

Some comments:

- In the callback we get the bounding box for the cell as arguments

---

- We adjust the position and height/width so that the widget is centered in the cell

The next callback is the signal strength widget, and we construct that as follows

```
void
cb_draw_signal_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                      size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                      HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL wwidth = 35;
    const HPDF_REAL wheight = 20;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 4;
    const size_t num_segments = 5;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_RED;
    // This should in reality be retrieved programmatically (for example from a DB)
    const size_t num_on_segments = 3;
    hpdftbl_widget_strength_meter(doc, page, wxpos, wypos, wwidth, wheight,
                                  num_segments, on_color, num_on_segments);
}
```

Some comments:

- In the callback we get the bounding box for the cell as arguments

- We adjust the position and height/width so that the widget is centered in the cell

With these callbacks it is now straightforward to construct the table with as follows

```
void
create_table_ex14(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex14: 2x2 table widget callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    // Use one label callback for the entire table
    hpdftbl_set_label_cb(tbl, cb_labels);
    // Name in top left corner
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_device_name);
    // Date in top right corner
    hpdftbl_set_cell_content_cb(tbl, 0, 1, cb_date);
    // Draw battery strength
    hpdftbl_set_cell_canvas_cb(tbl, 1, 0, cb_draw_battery_widget);
    // Draw signal strength
    hpdftbl_set_cell_canvas_cb(tbl, 1, 1, cb_draw_signal_widget);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

Some comments:

- For brevity, we have not shown the label and other content callback.

- The complete code is available as tut_ex14.c

Chapter 11

Serializing table data structures

A table and heme can be serialized to a JSON structure in a string or file and then read back. The serialization is a complete representation of the table and the theme. However, there is one crucial caveat. If any callback functions is used in the table they can not be serialized since they are represented by the address to the specific function.

However, this is a situation where a dynamic callback function can be used. They way this works is that the callback function is specified by name with one if the dynamic callback functions (e.g. hpdftbl_set_content_dyncb()). Such a callback will then be serialized as the name of the function.

When the table is de-serialized back into a table using the function hpdftbl_load() the functions must be available in some linked images to be resolved.

Please note that the table needs to be stroked (via for example hpdftbl_stroke()) before it can be serialized since a number of calculations of internal positions are not calculated until one of the stroke functions are called.

Note

> Some discussion can be held whether Yaml or Json should be used as serializing format. The reason for choosing Json was primary based on a) easier to manually manipulate a file where invisible spaces doesn't have grammatical meaning. b) existence of efficient libraries

11.1   Serializing a table to file

After the table have been stroked it can be saved to a file as the following snippet shows.
```
hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
hpdftbl_dump(tbl, "table_serialized.json");
```

The snippet above will write a JSON representation of the table to the file `table_serialized.json`. The full path is given and it is an error if some intermediate directory does not exist.

An example of a json file can be found here: `tut_ex40.json`

11.2   Serializing a table to a string buffer

This is done with the hpdftbl_dumps() function as the following example shows.
```
const size_t buffsize=100*1024;
char *sbuff=calloc(buffsize, sizeof(char));
hpdftbl_dumps(theme, sbuff, buffsize);
fprintf(stdout,"%s\n",sbuff);
free(sbuff);
```

11.3 Reading back a serialized table

The following snippet shows how the previously serialzed table can be read back and stroked to a PDF file

```
if( 0 == hpdftbl_load(&tbl, "table_serialized.json")  ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

It can be noted that we use the alternative stroke function hpdftbl_stroke_pos() which is used to stroke a table when the position is set within the table itself and don't need to be specified as arguments to the stroke functio as is necessary with hpdftbl_stroke().

Note

> An error check should always be performed when reading back a table since it is possible that the data have been corrupted.

11.4 Serializing a theme to a file

A theme can be serialized with the help of hpdftbl_theme_dump() as the following example that serializes the default theme

```
hpdftbl_theme_t *theme = hpdftbl_get_default_theme();
hpdftbl_theme_dump(theme,"out/default_theme.json");
```

11.5 Serializing a theme to a string buffer

This is done with the hpdftbl_theme_dumps() function as the following example shows

```
const size_t buffsize=2*1024;
char *sbuff=calloc(buffsize, sizeof(char));
hpdftbl_theme_t *theme = hpdftbl_get_default_theme();
hpdftbl_theme_dumps(theme, sbuff, buffsize);
fprintf(stdout,"%s\n",sbuff);
free(sbuff);
```

11.6 Reading back a serialized theme

A theme can be read back with the hpdftbl_theme_load() and hpdftbl_theme_loads() functions.

11.7 Example of reading back serialized theme and table

The following shows an example of creating a table with a theme where both theme and table are read back from previous serialized representation.

```
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
hpdftbl_theme_t theme;
if(0 == hpdftbl_load(tbl, "tests/tut_ex41.json")  ) {
    if(0 == hpdftbl_theme_load(&theme, "tests/tut41_theme.json")) {
        hpdftbl_apply_theme(tbl, &theme);
        hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
    } else {
        fprintf(stderr,"%s\n","Failed to load 'tests/default_theme.json'\n");
        exit(1);
    }
} else {
    fprintf(stderr,"%s\n","Failed to load 'tests/tut_ex41.json'\n");
    exit(1);
}
```

Chapter 12

API Overview

12.1 Table creation related functions

These calls relate to the creation, destruction and stroking of the table on the PDF page.

- hpdftbl_create() *Create a handle for a new table.*

- hpdftbl_create_title() *Create a handle for a new with a title.*

- hpdftbl_destroy() *Destroy (return) memory used by a table.*

- hpdftbl_stroke() *Stroke a table on the specified PDF page.*

- hpdftbl_setpos() *Set the size and position of the table.*

- hpdftbl_stroke_from_data() *Construct and stroke a table defined as a data structure.*

- hpdftbl_stroke_pos() *Create a handle for a new table using the position in the table structure.*

- hpdftbl_get_last_auto_height() *Get the height of the last table stroked.*

- hpdftbl_set_anchor_top_left() *Switch the anchor point of a table between top left and bottom left corner.*

- hpdftbl_get_anchor_top_left() *Get the current anchor point of table.*

12.2 Table error handling

- hpdftbl_set_errhandler() *Set and error handler callback.*

- hpdftbl_get_errstr() *Translate an error code into a human readable string.*

- hpdftbl_get_last_errcode() *Get the error code from last error raised*

- hpdftbl_default_table_error_handler() *A default error handler callback that print error to stdout and quits the process.*

- hpdftbl_get_last_errcode() *Return the last error code*

- hpdftbl_get_last_err_file() *Return the last filename and line number for where the last error occurred*

12.3 Theme handling methods

Themes is a technique to easier specify the look and feel to be re-used for multiple tables.

- hpdftbl_apply_theme() *Use the specified theme for look & feel of table*

- hpdftbl_get_default_theme() *Get the default theme. A good way to start and then modify.*

- hpdftbl_destroy_theme() *Free all memory structures used by a theme.*

12.4 Table layout adjusting functions

Adjusting the structure of the table (apart from number of rows and columns)

- hpdftbl_set_colwidth_percent() *Set the column width as a percentage of the entire table width.*

- hpdftbl_set_min_rowheight() *Specify the minimum row height in points*

- hpdftbl_set_bottom_vmargin_factor() *Specify the bottom margin for content as a fraction of the specified fontsize*

- hpdftbl_set_cellspan() *Define a cell to span multiple rows and columns.*

- hpdftbl_clear_spanning() *Remove all previous set cell spanning.*

12.5 Table style modifying functions

These functions are all about look and feel of the table.

- hpdftbl_use_labels() *Use labels in each cell.*

- hpdftbl_use_labelgrid() *Use shorter left gridlines that only goes down and cover labels*

- hpdftbl_set_background() *Set cell background color.*

- hpdftbl_set_outer_grid_style() *Set style of the table outer grid lines.*

- hpdftbl_set_inner_grid_style() *Set the style of both vertical and horizontal inner grid lines.*

- hpdftbl_set_inner_vgrid_style() *Set the style of table inner vertical grid lines.*

- hpdftbl_set_inner_hgrid_style() *Set the style of table inner horizontal grid lines.*

- hpdftbl_set_header_style() *Set the style for the table header row.*

- hpdftbl_set_header_halign() *Set the horizontal alignment of the header row.*

- hpdftbl_set_title_halign() *Set horizontal alignment for title.*

- hpdftbl_use_header() *Make the top row a header.*

- hpdftbl_set_label_style() *Set style for cell labels.*

- hpdftbl_set_row_content_style() *Set the content style for an entire row.*

- hpdftbl_set_col_content_style() *Set the content style for an entire column.*

- hpdftbl_set_content_style() *Set the content style for the entire table.*

- hpdftbl_set_cell_content_style() *Set the style for specified cell. This overrides andy style on the table level.*

- hpdftbl_set_title_style() *Set the style for the table title.*

12.6 Content handling

Content in a table can be specified in three ways

1. Manually for each cell by calling the hpdftbl_set_cell() function

2. In one go by creating a 1D data array for all cell

3. Creating a callback which returns the wanted value

- hpdftbl_set_cell() *Set content text in specified cell.*

- hpdftbl_set_tag() *Set the table tag. The tag is a* `void *` *an can be anything. The tag is the first parameter of all callbacks.*

- hpdftbl_set_title() *Set title text of table.*

- hpdftbl_set_labels() *Set label texts for the table from 1D-data array.*

- hpdftbl_set_content() *Set the content text for the entire table from a 1D-data array.*

12.7 Callback handling

Callbacks can be specified on both table but also on cell level. The simple rule is that if a cell has a callback that is used, otherwise the table callback is used.

- hpdftbl_set_content_cb() *Set table content callback.*

- hpdftbl_set_cell_content_cb() *Set cell content callback.*

- hpdftbl_set_cell_content_style_cb() *Set the cell style callback.*

- hpdftbl_set_content_style_cb() *Set the table style callback.*

- hpdftbl_set_label_cb() *Set table label callback.*

- hpdftbl_set_cell_label_cb() *Set the cell label callback.*

- hpdftbl_set_canvas_cb() *Set table canvas callback.*

- hpdftbl_set_cell_canvas_cb() *Set the cell canvas callback.*

- hpdftbl_set_post_cb() *Set the table post callback.*

12.8 Dynamic (late binding) callback handling

These are callbacks which set a function at runtime to be used as callback. This is useful when specifying the table for example as a structure stored in a database or in a file. The callback function is then specified as a string (the name of the callback function which is then resolved at runtime.

- hpdftbl_set_dlhandle()
  ∗Set the dynamic library load handle as returned by dlopen() or one of the predefined handles. By default, the handle is set to the predefined handle RTLD_DEFAULT. See man dlsym. This handle will control how the search for the name of the function will be conducted. The default will find any functions defined in any images linked and any libraries linked at compile time. It will **not** find functions defined in libraries that are dynamically loaded. In that case you shoult specify the handle returned by dlopen().

- hpdftbl_set_content_dyncb() *Set the name for the table content callback.*

- hpdftbl_set_cell_content_dyncb() *Set the name for the cell content callback.*

- hpdftbl_set_label_dyncb()

- hpdftbl_set_cell_label_dyncb() *Set the name for the cell label content callback.*

- hpdftbl_set_content_style_dyncb() *Set the name for the table content style callback.*

- hpdftbl_set_cell_content_style_dyncb() *Set the name for the cell content style callback.*

- hpdftbl_set_cell_canvas_dyncb() *Set the name for the cell canvas callback.*

- hpdftbl_set_post_dyncb() *Set the name for the table post callback.*

12.9 Serializing

- hpdftbl_dump() *Export table in json format to named file.*

- hpdftbl_dumps() *Export table in json format to sting buffer.*

- hpdftbl_load() *Import table in json format from named file.*

- hpdftbl_loads() *Import table in json format from string buffer.*

- hpdftbl_theme_dump() *Export theme in json format to named file.*

- hpdftbl_theme_dumps() *Export theme in json format to sting buffer.*

- hpdftbl_theme_load() *Import theme in json format from named file.*

- hpdftbl_theme_loads() *Import theme in json format from string buffer.*

12.10 Text encoding

- hpdftbl_set_text_encoding() *Specify text encodation to use.*

- hpdftbl_encoding_text_out() *Stroke a text with current encoding.*

12.11 Misc utility function

- HPDF_RoundedCornerRectangle() *Draw a rectangle with rounded corners.*

- hpdftbl_stroke_grid() ∗Stroke a grid on the PDF page (entire page). This is useful to position the table on a page. The grid is measured in points i.e. postscript natural units.

Chapter 13

Data Structure Index

13.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 14

File Index

14.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 15

Data Structure Documentation

15.1 grid_style Struct Reference

Specification for table grid lines.

```
#include <hpdftbl.h>
```

Data Fields

- HPDF_REAL width
- HPDF_RGBColor color
- hpdftbl_line_dashstyle_t line_dashstyle

15.1.1 Detailed Description

Specification for table grid lines.

Contains line properties used when stroking a grid line

15.1.2 Field Documentation

15.1.2.1 color

```
HPDF_RGBColor color
```

Color of grids

Referenced by hpdftbl_apply_theme(), hpdftbl_set_inner_hgrid_style(), hpdftbl_set_inner_tgrid_style(), hpdftbl_set_inner_vgrid_style(and hpdftbl_set_outer_grid_style().

15.1.2.2 line_dashstyle

`hpdftbl_line_dashstyle_t line_dashstyle`

Line style for grid

Referenced by hpdftbl_apply_theme(), hpdftbl_set_inner_hgrid_style(), hpdftbl_set_inner_tgrid_style(), hpdftbl_set_inner_vgrid_style(), and hpdftbl_set_outer_grid_style().

15.1.2.3 width

`HPDF_REAL width`

Line width of grids

Referenced by hpdftbl_apply_theme(), hpdftbl_set_inner_hgrid_style(), hpdftbl_set_inner_tgrid_style(), hpdftbl_set_inner_vgrid_style(), and hpdftbl_set_outer_grid_style().

The documentation for this struct was generated from the following file:

- hpdftbl.h

15.2 hpdftbl Struct Reference

Core table handle.

`#include <hpdftbl.h>`

Data Fields

- HPDF_Doc pdf_doc
- HPDF_Page pdf_page
- size_t cols
- size_t rows
- HPDF_REAL posx
- HPDF_REAL posy
- HPDF_REAL height
- HPDF_REAL minrowheight
- _Bool anchor_is_top_left
- HPDF_REAL bottom_vmargin_factor
- HPDF_REAL width
- void ∗ tag
- char ∗ title_txt
- hpdf_text_style_t title_style
- hpdf_text_style_t header_style
- _Bool use_header_row
- hpdf_text_style_t label_style
- _Bool use_cell_labels

- _Bool use_label_grid_style
- hpdf_text_style_t content_style
- hpdftbl_content_callback_t label_cb
- char * label_dyncb
- hpdftbl_content_callback_t content_cb
- char * content_dyncb
- hpdftbl_content_style_callback_t content_style_cb
- char * content_style_dyncb
- hpdftbl_canvas_callback_t canvas_cb
- char * canvas_dyncb
- hpdftbl_callback_t post_cb
- char * post_dyncb
- hpdftbl_grid_style_t outer_grid
- hpdftbl_grid_style_t inner_vgrid
- hpdftbl_grid_style_t inner_hgrid
- hpdftbl_grid_style_t inner_tgrid
- _Bool use_zebra
- int zebra_phase
- HPDF_RGBColor zebra_color1
- HPDF_RGBColor zebra_color2
- float * col_width_percent
- hpdftbl_cell_t * cells

15.2.1 Detailed Description

Core table handle.

This is the main structure that contains all information for the table. The basic structure is an array of cells.

See also

hpdftbl_cell_t

Examples

example01.c, tut_ex00.c, tut_ex01.c, tut_ex02.c, tut_ex02_1.c, tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex09.c, tut_ex10.c, tut_ex11.c, tut_ex12.c, tut_ex14.c, tut_ex15.c, tut_ex15_1.c, tut_ex20.c, tut_ex30.c, tut_ex40.c, and tut_ex41.c.

15.2.2 Field Documentation

15.2.2.1 anchor_is_top_left

```
_Bool anchor_is_top_left
```

Is the table anchor to be upper top left or bottom left

Referenced by hpdftbl_create_title(), hpdftbl_get_anchor_top_left(), and hpdftbl_set_anchor_top_left().

15.2.2.2 bottom_vmargin_factor

```
HPDF_REAL bottom_vmargin_factor
```

The content text bottom margin as a factor of the fontsize

Referenced by hpdftbl_get_theme(), and hpdftbl_set_bottom_vmargin_factor().

15.2.2.3 canvas_cb

```
hpdftbl_canvas_callback_t canvas_cb
```

Table canvas callback. Will be called for each cell unless the cell has its own canvas callback

Referenced by hpdftbl_set_canvas_cb().

15.2.2.4 canvas_dyncb

```
char* canvas_dyncb
```

Table canvas dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

Referenced by hpdftbl_destroy().

15.2.2.5 cells

```
hpdftbl_cell_t* cells
```

Reference to all an array of cells in the table

Referenced by hpdftbl_clear_spanning(), hpdftbl_create_title(), hpdftbl_set_cell(), hpdftbl_set_cell_canvas_cb(), hpdftbl_set_cell_content_cb(), hpdftbl_set_cell_content_style(), hpdftbl_set_cell_content_style_cb(), hpdftbl_set_cell_label_cb(), hpdftbl_set_cellspan(), hpdftbl_set_content(), and hpdftbl_set_labels().

15.2.2.6 col_width_percent

```
float* col_width_percent
```

User specified column width array as fraction of the table width. Defaults to equ-width

Referenced by hpdftbl_create_title(), hpdftbl_destroy(), and hpdftbl_set_colwidth_percent().

15.2.2.7 cols

```
size_t cols
```

Number of columns in table

Referenced by hpdftbl_clear_spanning(), hpdftbl_create_title(), hpdftbl_destroy(), hpdftbl_set_cellspan(), hpdftbl_set_colwidth_percent(), hpdftbl_set_content(), hpdftbl_set_labels(), and hpdftbl_set_row_content_style().

15.2.2.8 content_cb

```
hpdftbl_content_callback_t content_cb
```

Table content callback. Will be called for each cell unless the cell has its own content callback

Referenced by hpdftbl_set_content_cb().

15.2.2.9 content_dyncb

```
char* content_dyncb
```

Table content dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

Referenced by hpdftbl_destroy().

15.2.2.10 content_style

```
hpdf_text_style_t content_style
```

Content style

Referenced by hpdftbl_get_theme(), hpdftbl_set_background(), and hpdftbl_set_content_style().

15.2.2.11 content_style_cb

```
hpdftbl_content_style_callback_t content_style_cb
```

Style for content callback. Will be called for each cell unless the cell has its own content style callback

Referenced by hpdftbl_set_content_style_cb().

15.2.2.12 content_style_dyncb

`char* content_style_dyncb`

Table content style dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

Referenced by hpdftbl_destroy().

15.2.2.13 header_style

`hpdf_text_style_t header_style`

Header style

Referenced by hpdftbl_get_theme(), hpdftbl_set_header_halign(), and hpdftbl_set_header_style().

15.2.2.14 height

`HPDF_REAL height`

Table height. If specified as 0 then the height will be automatically calculated

Referenced by hpdftbl_setpos(), and hpdftbl_stroke_pos().

15.2.2.15 inner_hgrid

`hpdftbl_grid_style_t inner_hgrid`

Table inner horizontal border settings, if width>0 this takes precedence over the generic inner border

Referenced by hpdftbl_get_theme(), and hpdftbl_set_inner_hgrid_style().

15.2.2.16 inner_tgrid

`hpdftbl_grid_style_t inner_tgrid`

Table inner horizontal top border settings, if width>0 this takes precedence over the generic horizontal and inner horizontal border

Referenced by hpdftbl_get_theme(), and hpdftbl_set_inner_tgrid_style().

15.2.2.17 inner_vgrid

hpdftbl_grid_style_t inner_vgrid

Table inner vertical border settings, if width>0 this takes precedence over the generic inner border

Referenced by hpdftbl_get_theme(), and hpdftbl_set_inner_vgrid_style().

15.2.2.18 label_cb

hpdftbl_content_callback_t label_cb

Table content callback. Will be called for each cell unless the cella has its own content callback

Referenced by hpdftbl_set_label_cb().

15.2.2.19 label_dyncb

char* label_dyncb

Table label dynamic callback name. The name is created vi strdup() and must be freed on destruction

Referenced by hpdftbl_destroy().

15.2.2.20 label_style

hpdf_text_style_t label_style

Label style settings

Referenced by hpdftbl_get_theme(), and hpdftbl_set_label_style().

15.2.2.21 minrowheight

HPDF_REAL minrowheight

Minimum table row height. If specified as 0 it has no effect

Referenced by hpdftbl_set_min_rowheight().

15.2.2.22 outer_grid

`hpdftbl_grid_style_t` outer_grid

Table outer border settings

Referenced by hpdftbl_get_theme(), and hpdftbl_set_outer_grid_style().

15.2.2.23 pdf_doc

`HPDF_Doc pdf_doc`

PDF document references

15.2.2.24 pdf_page

`HPDF_Page pdf_page`

PDF page reference

Referenced by hpdftbl_set_line_dash().

15.2.2.25 post_cb

`hpdftbl_callback_t` post_cb

Post table creation callback. This is an opportunity for a client to do any special table manipulation before the table is stroked to the page. A reference to the table will be passed on in the callback.

Referenced by hpdftbl_set_post_cb().

15.2.2.26 post_dyncb

`char* post_dyncb`

Table post dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

Referenced by hpdftbl_destroy().

15.2.2.27 posx

`HPDF_REAL posx`

X-position of table. Reference point defaults to lower left but can be changed by calling hpdftbl_set_anchor_top_left()

Referenced by hpdftbl_setpos(), and hpdftbl_stroke_pos().

15.2.2.28 posy

`HPDF_REAL posy`

Y-position of table. Reference point defaults to lower left but can be changed by calling hpdftbl_set_anchor_top_left()

Referenced by hpdftbl_setpos(), and hpdftbl_stroke_pos().

15.2.2.29 rows

`size_t rows`

Number of rows in table

Referenced by hpdftbl_clear_spanning(), hpdftbl_create_title(), hpdftbl_destroy(), hpdftbl_set_cellspan(), hpdftbl_set_col_content_style(), hpdftbl_set_content(), and hpdftbl_set_labels().

15.2.2.30 tag

`void* tag`

Optional tag used in callbacks. This can be used to identify the table or add any reference needed by a particular application

Referenced by hpdftbl_set_tag().

15.2.2.31 title_style

`hpdf_text_style_t title_style`

Title style

Referenced by hpdftbl_get_theme(), hpdftbl_set_title_halign(), and hpdftbl_set_title_style().

15.2.2.32   title_txt

`char* title_txt`

Title text

Referenced by [hpdftbl_create_title()](), [hpdftbl_destroy()](), and [hpdftbl_set_title()]().

15.2.2.33   use_cell_labels

`_Bool use_cell_labels`

Flag to determine if cell labels should be used

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_theme()](), and [hpdftbl_use_labels()]().

15.2.2.34   use_header_row

`_Bool use_header_row`

Flag to determine if the first row in the table should be formatted as a header row

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_theme()](), and [hpdftbl_use_header()]().

15.2.2.35   use_label_grid_style

`_Bool use_label_grid_style`

Flag to determine of the short vertical label border should be used. Default is to use half grid.

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_theme()](), [hpdftbl_use_labelgrid()](), and [hpdftbl_use_labels()]().

15.2.2.36   use_zebra

`_Bool use_zebra`

Use alternating background color on every second line TRUE or FALSE. Defaults to FALSE.

See also

> [hpdftbl_set_zebra()]()

Referenced by [hpdftbl_get_theme()](), and [hpdftbl_set_zebra()]().

15.2.2.37 width

`HPDF_REAL width`

Table width

Referenced by hpdftbl_setpos(), and hpdftbl_stroke_pos().

15.2.2.38 zebra_color1

`HPDF_RGBColor zebra_color1`

First zebra color.

See also

> hpdftbl_set_zebra_color()

Referenced by hpdftbl_get_theme(), and hpdftbl_set_zebra_color().

15.2.2.39 zebra_color2

`HPDF_RGBColor zebra_color2`

Second zebra color.

See also

> hpdftbl_set_zebra_color()

Referenced by hpdftbl_get_theme(), and hpdftbl_set_zebra_color().

15.2.2.40 zebra_phase

`int zebra_phase`

Determine if we start with color1 (phase=0) or start with color2 (phase=1)

See also

> hpdftbl_set_zebra()

Referenced by hpdftbl_get_theme(), and hpdftbl_set_zebra().

The documentation for this struct was generated from the following file:

- hpdftbl.h

15.3 hpdftbl_cell Struct Reference

Specification of individual cells in the table.

```
#include <hpdftbl.h>
```

Data Fields

- size_t row
- size_t col
- char ∗ label
- char ∗ content
- size_t colspan
- size_t rowspan
- HPDF_REAL height
- HPDF_REAL width
- HPDF_REAL delta_x
- HPDF_REAL delta_y
- HPDF_REAL textwidth
- hpdftbl_content_callback_t content_cb
- char ∗ content_dyncb
- hpdftbl_content_callback_t label_cb
- char ∗ label_dyncb
- hpdftbl_content_style_callback_t style_cb
- char ∗ content_style_dyncb
- hpdftbl_canvas_callback_t canvas_cb
- char ∗ canvas_dyncb
- hpdf_text_style_t content_style
- struct hpdftbl_cell ∗ parent_cell

15.3.1 Detailed Description

Specification of individual cells in the table.

This structure contains all information pertaining to each cell in the table. The position of the cell is given as relative position from the lower left corner of the table.

15.3.2 Field Documentation

15.3.2.1 canvas_cb

hpdftbl_canvas_callback_t canvas_cb

Canvas callback. If this is specified then this will override any canvas callback specified for the table

Referenced by hpdftbl_set_cell_canvas_cb().

15.3.2.2 canvas_dyncb

```
char* canvas_dyncb
```

Cell canvas dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

15.3.2.3 col

```
size_t col
```

When serializing it makes it easier to have row,col in each cell

Referenced by hpdftbl_create_title().

15.3.2.4 colspan

```
size_t colspan
```

Number of column this cell spans

Referenced by hpdftbl_clear_spanning(), hpdftbl_set_cell(), and hpdftbl_set_cellspan().

15.3.2.5 content

```
char* content
```

String reference for cell content

Referenced by hpdftbl_set_cell(), and hpdftbl_set_content().

15.3.2.6 content_cb

```
hpdftbl_content_callback_t content_cb
```

Content callback. If this is specified then this will override any content callback specified for the table

Referenced by hpdftbl_set_cell_content_cb().

15.3.2.7 content_dyncb

```
char* content_dyncb
```

Cell content dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

15.3.2.8 content_style

[hpdf_text_style_t](#) content_style

The style of the text content. If a style callback is specified the callback will override this setting

Referenced by [hpdftbl_set_cell_content_style()](#).

15.3.2.9 content_style_dyncb

char* content_style_dyncb

Cell content style dynamic callback name. The name is created vi strdup() and must be freed on destruction

15.3.2.10 delta_x

HPDF_REAL delta_x

X-Position of cell from bottom left of table

15.3.2.11 delta_y

HPDF_REAL delta_y

Y-Position of cell from bottom left of table

15.3.2.12 height

HPDF_REAL height

Height of cell

15.3.2.13 label

char* label

String reference for label text

Referenced by [hpdftbl_set_cell()](#), and [hpdftbl_set_labels()](#).

15.3.2.14 label_cb

`hpdftbl_content_callback_t` `label_cb`

Label callback. If this is specified then this will override any content callback specified for the table

Referenced by hpdftbl_set_cell_label_cb().

15.3.2.15 label_dyncb

`char* label_dyncb`

Cell label dynamic callback name. The name is created vi `strdup()` and must be freed on destruction

15.3.2.16 parent_cell

`struct hpdftbl_cell* parent_cell`

Parent cell. If this cell is part of another cells row or column spanning this is a reference to this parent cell. Normal cells without spanning has NULL as parent cell.

Referenced by hpdftbl_clear_spanning(), hpdftbl_set_cell(), hpdftbl_set_cell_canvas_cb(), hpdftbl_set_cell_content_cb(), hpdftbl_set_cell_content_style_cb(), hpdftbl_set_cell_label_cb(), and hpdftbl_set_cellspan().

15.3.2.17 row

`size_t row`

When serializing it makes it easier to have row,col in each cell

Referenced by hpdftbl_create_title().

15.3.2.18 rowspan

`size_t rowspan`

Number of rows this cell spans

Referenced by hpdftbl_clear_spanning(), hpdftbl_set_cell(), and hpdftbl_set_cellspan().

15.3.2.19 style_cb

[hpdftbl_content_style_callback_t](#) style_cb

Style for content callback. If this is specified then this will override any style content callback specified for the table

Referenced by [hpdftbl_set_cell_content_style_cb()](#).

15.3.2.20 textwidth

HPDF_REAL textwidth

Width of content string

15.3.2.21 width

HPDF_REAL width

Width of cells

The documentation for this struct was generated from the following file:

- [hpdftbl.h](#)

15.4 hpdftbl_cell_spec Struct Reference

Used in data driven table creation.

#include <hpdftbl.h>

Data Fields

- size_t [row](#)
- size_t [col](#)
- unsigned [rowspan](#)
- unsigned [colspan](#)
- char ∗ [label](#)
- [hpdftbl_content_callback_t content_cb](#)
- [hpdftbl_content_callback_t label_cb](#)
- [hpdftbl_content_style_callback_t style_cb](#)
- [hpdftbl_canvas_callback_t canvas_cb](#)

15.4.1 Detailed Description

Used in data driven table creation.

A table can be specified by creating a array of this structure together with the hpdftbl_spec_t structure. The array should have one entry for each cell in the table.

See also

hpdftbl_stroke_from_data()

Examples

example01.c, and tut_ex13_2.c.

15.4.2 Field Documentation

15.4.2.1 canvas_cb

hpdftbl_canvas_callback_t canvas_cb

Canvas callback for this cell

Referenced by hpdftbl_stroke_from_data().

15.4.2.2 col

size_t col

Row for specified cell

Referenced by hpdftbl_stroke_from_data().

15.4.2.3 colspan

unsigned colspan

Number of columns the specified cell should span

Referenced by hpdftbl_stroke_from_data().

15.4.2.4 content_cb

hpdftbl_content_callback_t content_cb

Content callback for this cell

Referenced by hpdftbl_stroke_from_data().

15.4.2.5 label

char* label

The label for this cell

Referenced by hpdftbl_stroke_from_data().

15.4.2.6 label_cb

hpdftbl_content_callback_t label_cb

Label callback for this cell

Referenced by hpdftbl_stroke_from_data().

15.4.2.7 row

size_t row

Row for specified cell

Examples

tut_ex13_2.c.

Referenced by hpdftbl_stroke_from_data().

15.4.2.8 rowspan

unsigned rowspan

Number of rows the specified cell should span

Referenced by hpdftbl_stroke_from_data().

15.4.2.9 style_cb

`hpdftbl_content_style_callback_t` style_cb

Content style callback for this cell

Referenced by hpdftbl_stroke_from_data().

The documentation for this struct was generated from the following file:

- hpdftbl.h

15.5 hpdftbl_errcode_entry Struct Reference

An entry in the error string table.

Data Fields

- char ∗ errstr
- unsigned errcode

15.5.1 Detailed Description

An entry in the error string table.

15.5.2 Field Documentation

15.5.2.1 errcode

`unsigned errcode`

The error code from HPDF library

15.5.2.2 errstr

`char* errstr`

Pointer to the error string

Referenced by hpdftbl_get_last_errcode().

The documentation for this struct was generated from the following file:

- hpdftbl_errstr.c

15.6 hpdftbl_spec Struct Reference

Used in data driven table creation.

```
#include <hpdftbl.h>
```

Data Fields

- char ∗ title
- _Bool use_header
- _Bool use_labels
- _Bool use_labelgrid
- size_t rows
- size_t cols
- HPDF_REAL xpos
- HPDF_REAL ypos
- HPDF_REAL width
- HPDF_REAL height
- hpdftbl_content_callback_t content_cb
- hpdftbl_content_callback_t label_cb
- hpdftbl_content_style_callback_t style_cb
- hpdftbl_callback_t post_cb
- hpdftbl_cell_spec_t ∗ cell_spec

15.6.1 Detailed Description

Used in data driven table creation.

This is used together with an array of cell specification hpdftbl_cell_spec_t to specify the layout of a table.

Examples

example01.c, tut_ex13_1.c, and tut_ex13_2.c.

15.6.2 Field Documentation

15.6.2.1 cell_spec

```
hpdftbl_cell_spec_t* cell_spec
```

Array of cell specification

Referenced by hpdftbl_stroke_from_data().

15.6.2.2   cols

```
size_t cols
```

Number of columns in the table

Referenced by hpdftbl_stroke_from_data().

15.6.2.3   content_cb

```
hpdftbl_content_callback_t content_cb
```

Content callback for this table

Referenced by hpdftbl_stroke_from_data().

15.6.2.4   height

```
HPDF_REAL height
```

Height of table

Referenced by hpdftbl_stroke_from_data().

15.6.2.5   label_cb

```
hpdftbl_content_callback_t label_cb
```

Label callback for this table

Referenced by hpdftbl_stroke_from_data().

15.6.2.6   post_cb

```
hpdftbl_callback_t post_cb
```

Post table creation callback.

Referenced by hpdftbl_stroke_from_data().

15.6.2.7 rows

```
size_t rows
```

Number of rows in the table

Referenced by [hpdftbl_stroke_from_data()](#).

15.6.2.8 style_cb

```
hpdftbl_content_style_callback_t style_cb
```

Content style callback for table

Referenced by [hpdftbl_stroke_from_data()](#).

15.6.2.9 title

```
char* title
```

Table title

Examples

[example01.c](#), [tut_ex13_1.c](#), and [tut_ex13_2.c](#).

Referenced by [hpdftbl_stroke_from_data()](#).

15.6.2.10 use_header

```
_Bool use_header
```

Use a header for the table

Referenced by [hpdftbl_stroke_from_data()](#).

15.6.2.11 use_labelgrid

```
_Bool use_labelgrid
```

Use label grid in table

Referenced by [hpdftbl_stroke_from_data()](#).

15.6.2.12 use_labels

`_Bool use_labels`

Use labels in table

Referenced by hpdftbl_stroke_from_data().

15.6.2.13 width

`HPDF_REAL width`

Width of table

Referenced by hpdftbl_stroke_from_data().

15.6.2.14 xpos

`HPDF_REAL xpos`

X-position for table

Referenced by hpdftbl_stroke_from_data().

15.6.2.15 ypos

`HPDF_REAL ypos`

Y-position for table

Referenced by hpdftbl_stroke_from_data().

The documentation for this struct was generated from the following file:

- hpdftbl.h

15.7 hpdftbl_theme Struct Reference

Define a set of styles into a table theme.

`#include <hpdftbl.h>`

Data Fields

- hpdf_text_style_t content_style
- hpdf_text_style_t label_style
- hpdf_text_style_t header_style
- hpdf_text_style_t title_style
- hpdftbl_grid_style_t outer_border
- _Bool use_labels
- _Bool use_label_grid_style
- _Bool use_header_row
- hpdftbl_grid_style_t inner_vborder
- hpdftbl_grid_style_t inner_hborder
- hpdftbl_grid_style_t inner_tborder
- _Bool use_zebra
- int zebra_phase
- HPDF_RGBColor zebra_color1
- HPDF_RGBColor zebra_color2
- HPDF_REAL bottom_vmargin_factor

15.7.1 Detailed Description

Define a set of styles into a table theme.

Contains all information about the styles of various elements in the table that together make up the table style

Examples

example01.c, and tut_ex41.c.

15.7.2 Field Documentation

15.7.2.1 bottom_vmargin_factor

```
HPDF_REAL bottom_vmargin_factor
```

Specify the vertical margin factor

Referenced by hpdftbl_apply_theme(), hpdftbl_get_default_theme(), and hpdftbl_get_theme().

15.7.2.2 content_style

```
hpdf_text_style_t content_style
```

Content text style

Referenced by hpdftbl_apply_theme(), hpdftbl_get_default_theme(), and hpdftbl_get_theme().

15.7.2.3 header_style

`hpdf_text_style_t` `header_style`

Header text style

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_default_theme()](), and [hpdftbl_get_theme()]().

15.7.2.4 inner_hborder

`hpdftbl_grid_style_t` `inner_hborder`

Table inner horizontal border settings, if width>0 this takes precedence over the generic inner border

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_default_theme()](), and [hpdftbl_get_theme()]().

15.7.2.5 inner_tborder

`hpdftbl_grid_style_t` `inner_tborder`

Table inner horizontal top border settings, if width>0 this takes precedence over the generic horizontal and inner horizontal border

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_default_theme()](), and [hpdftbl_get_theme()]().

15.7.2.6 inner_vborder

`hpdftbl_grid_style_t` `inner_vborder`

Table inner vertical border settings, if width>0 this takes precedence over the generic inner border

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_default_theme()](), and [hpdftbl_get_theme()]().

15.7.2.7 label_style

`hpdf_text_style_t` `label_style`

Label text style

Referenced by [hpdftbl_apply_theme()](), [hpdftbl_get_default_theme()](), and [hpdftbl_get_theme()]().

15.7.2.8 outer_border

`hpdftbl_grid_style_t` outer_border

Table outer border style

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.9 title_style

`hpdf_text_style_t` title_style

Table title text style

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.10 use_header_row

`_Bool` use_header_row

Flag if header row should be used

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.11 use_label_grid_style

`_Bool` use_label_grid_style

Flag if the special short vertical grid style for labels should be used

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.12 use_labels

`_Bool` use_labels

Flag if cell labels should be used

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.13 use_zebra

`_Bool use_zebra`

Use alternating background color on every second line TRUE or FALSE. Defaults to FALSE.

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.14 zebra_color1

`HPDF_RGBColor zebra_color1`

First zebra color.

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.15 zebra_color2

`HPDF_RGBColor zebra_color2`

Second zebra color.

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

15.7.2.16 zebra_phase

`int zebra_phase`

Start with color1 or color2

Referenced by [hpdftbl_apply_theme()](#), [hpdftbl_get_default_theme()](#), and [hpdftbl_get_theme()](#).

The documentation for this struct was generated from the following file:

- [hpdftbl.h](#)

15.8   line_dash_style Struct Reference

Definition of a dashed line style.

Data Fields

- HPDF_UINT16 dash_ptn [8]
- size_t num

15.8.1 Detailed Description

Definition of a dashed line style.

15.8.2 Field Documentation

15.8.2.1 dash_ptn

```
HPDF_UINT16 dash_ptn[8]
```

HPDF dash line definition

15.8.2.2 num

```
size_t num
```

Number of segments in the dashed line

The documentation for this struct was generated from the following file:

- hpdftbl.c

15.9 text_style Struct Reference

Specification of a text style.

```
#include <hpdftbl.h>
```

Data Fields

- char ∗ font
- HPDF_REAL fsize
- HPDF_RGBColor color
- HPDF_RGBColor background
- hpdftbl_text_align_t halign

15.9.1 Detailed Description

Specification of a text style.

This structure collects the basic properties for a text string (font, color, background, horizontal alignment)

Examples

tut_ex09.c.

15.9.2 Field Documentation

15.9.2.1 background

`HPDF_RGBColor background`

Font background color

Examples

tut_ex09.c.

Referenced by hpdftbl_apply_theme(), hpdftbl_set_background(), hpdftbl_set_cell_content_style(), hpdftbl_set_content_style(), hpdftbl_set_header_style(), hpdftbl_set_label_style(), and hpdftbl_set_title_style().

15.9.2.2 color

`HPDF_RGBColor color`

Font color

Examples

tut_ex09.c.

Referenced by hpdftbl_apply_theme(), hpdftbl_set_cell_content_style(), hpdftbl_set_content_style(), hpdftbl_set_header_style(), hpdftbl_set_label_style(), and hpdftbl_set_title_style().

15.9.2.3 font

```
char* font
```

Font face name

Examples

> tut_ex09.c.

Referenced by hpdftbl_apply_theme(), hpdftbl_set_cell_content_style(), hpdftbl_set_content_style(), hpdftbl_set_header_style(), hpdftbl_set_label_style(), and hpdftbl_set_title_style().

15.9.2.4 fsize

```
HPDF_REAL fsize
```

Font size

Examples

> tut_ex09.c.

Referenced by hpdftbl_apply_theme(), hpdftbl_set_cell_content_style(), hpdftbl_set_content_style(), hpdftbl_set_header_style(), hpdftbl_set_label_style(), and hpdftbl_set_title_style().

15.9.2.5 halign

```
hpdftbl_text_align_t halign
```

Text horizontal alignment

Examples

> tut_ex09.c.

Referenced by hpdftbl_apply_theme(), hpdftbl_set_header_halign(), and hpdftbl_set_title_halign().

The documentation for this struct was generated from the following file:

- hpdftbl.h

Chapter 16

File Documentation

16.1 unit_test.inc.h File Reference

Common functions for all unit-test/examples.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <execinfo.h>
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#include <sys/utsname.h>
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#include "../src/hpdftbl.h"
```

Macros

- #define **TESTS_DIR** "./tests/"
- #define TUTEX_MAIN(_tbl_, _showgrid_)

  Macro to create a main() function to call the table creation function for each example. The name to the table function is given as the first and only argument to the macro.

Typedefs

- typedef char ∗∗ **content_t**

  An array of char pointers.

Functions

- void [setup_hpdf](HPDF_Doc *pdf_doc, HPDF_Page *pdf_page, _Bool addgrid)

    Create a new PDF document with one page in A4 format.
- char ∗ [setup_filename](int argc, char **argv)

    Return a pointer to a static buffer that holds the filename to be used for the PDF page.
- int [stroke_to_file](HPDF_Doc pdf_doc, int argc, char **argv)

    Stroke the created PDF page to a file.
- void [setup_dummy_content]([content_t](content_t *content, size_t rows, size_t cols)

    Create an array of char pointers to simulate real table data.
- void [setup_dummy_content_label]([content_t](content_t *content, [content_t](content_t *labels, size_t rows, size_t cols)

    Create both array of char pointers to simulate real table content as well as an array of simulated labels.
- char ∗ [mkfullpath](char *filename)

    Add the full path to the tests directory as prefix to the supplied filename as argument.

Variables

- _Bool [run_as_unit_test] = [FALSE]

    For the case when we use this example as a unit/integration test we do not want data such as dates, times, and system-information to be updated since the checks compare the result to a stored copy of the PDF file.
- jmp_buf [_hpdftbl_jmp_env]

    For simulated exception handling.

16.1.1 Detailed Description

Common functions for all unit-test/examples.

DO NOT EDIT [examples/unit_test.inc.h]. Generated from unit_test.inc.h.in by configure.

16.1.2 Macro Definition Documentation

16.1.2.1 TUTEX_MAIN

```
#define TUTEX_MAIN(
            _tbl_,
            _showgrid_ )
```

Value:
```
int \
main(int argc, char **argv) { \
    HPDF_Doc pdf_doc; \
    HPDF_Page pdf_page;    \
    run_as_unit_test = 2==argc ; \
    if (setjmp(_hpdftbl_jmp_env)) { \
        return EXIT_FAILURE; \
    } \
    hpdftbl_set_errhandler(table_error_handler); \
    setup_hpdf(&pdf_doc, &pdf_page, _showgrid_); \
    _tbl_(pdf_doc, pdf_page); \
    if( -1 == stroke_to_file(pdf_doc, argc, argv) ) \
        return EXIT_FAILURE; \
    else \
        return EXIT_SUCCESS; \
}
```

Macro to create a main() function to call the table creation function for each example. The name to the table function is given as the first and only argument to the macro.

Parameters

tbl	The name of the main table creation function to be called.
_←_ _showgrid←_ _	Set to TRUE to display a dot-grid on the paper, FALSE otherwise.

Examples

> tut_ex00.c, tut_ex01.c, tut_ex02.c, tut_ex02_1.c, tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex09.c, tut_ex10.c, tut_ex11.c, tut_ex12.c, tut_ex13_1.c, tut_ex13_2.c, tut_ex14.c, tut_ex15.c, tut_ex15_1.c, tut_ex20.c, tut_ex30.c, tut_ex40.c, and tut_ex41.c.

16.1.3 Function Documentation

16.1.3.1 mkfullpath()

```
char * mkfullpath (
            char * filename )
```

Add the full path to the tests directory as prefix to the supplied filename as argument.

This is needed since it is not possible at compile time to know the location of the tests directory due to the autotools feature of VPATH. This makes it possible to build in a arbitrary separate directory from the source. The autotools support the variable '.' which gets replaced with the actual src directory which makes it possible to find the tests directory.

The replacement happens when the autotools creates `unit_test.inc.h` from `unit_test.inc.h.in` with the created "config.status"

Parameters

filename	The base filename (relative to the tests directory)

Returns

> Pointer to a dynamic allocated storage for the full path filename. It is the clients responsibility to free() the allocated memory.

Examples

> tut_ex40.c, and tut_ex41.c.

16.1.3.2 setup_dummy_content()

```
void setup_dummy_content (
            content_t * content,
            size_t rows,
            size_t cols )
```

Create an array of char pointers to simulate real table data.

Parameters

out	*content*	A pointer to an array of char pointers
in	*rows*	Number of rows in table
in	*cols*	Number of columns in table

Examples

tut_ex02.c, tut_ex10.c, tut_ex11.c, tut_ex12.c, tut_ex15.c, and tut_ex15_1.c.

16.1.3.3 setup_dummy_content_label()

```
void setup_dummy_content_label (
            content_t * content,
            content_t * labels,
            size_t rows,
            size_t cols )
```

Create both array of char pointers to simulate real table content as well as an array of simulated labels.

Parameters

out	*content*	A pointer to an array of char pointers to represent content in a table
out	*labels*	A pointer to an array of char pointers to represent labels in a table
in	*rows*	Number of rows in table
in	*cols*	Number of columns in table

Examples

tut_ex00.c, tut_ex04.c, tut_ex05.c, tut_ex20.c, tut_ex40.c, and tut_ex41.c.

16.1.3.4 setup_filename()

```
char * setup_filename (
            int argc,
            char ** argv )
```

Return a pointer to a static buffer that holds the filename to be used for the PDF page.

The filename is create from the program arguments. If there are no arguments the file name will have the same basename as the program (but with an added "∗.pdf" suffix and path "out/".

If there are exactly on argument this will in its entirety (incl path) be taken as the full name for the file.

Warning

> It is the calling functions responsibility to check for a NULL return value.

More than one program argument will generate an error (return NULL)

Parameters

argc	The main() functions argc argument
argv	The main() functions argv argument

Returns

> NULL on error, otherwise a pointer to a static string that holds the filename.

Referenced by stroke_to_file().

16.1.3.5 setup_hpdf()

```
void setup_hpdf (
            HPDF_Doc * pdf_doc,
            HPDF_Page * pdf_page,
            _Bool addgrid )
```

Create a new PDF document with one page in A4 format.

This initializes a basic PDF document object and a page object that can then be used. A typical calling sequence would be:
```
HPDF_Doc pdf_doc;
HPDF_Page pdf_page;
setup_hpdf(&pdf_doc, &pdf_page, FALSE);
```

The pdf_doc and pdf_page can subsequently be used in the other PDF functions.

By setting the `addgrid` to TRUE the paper background will be gridlines with coordinate system units in points. This is very useful to precisely position text and graphics on a page.

Parameters

`out`	*pdf_doc*	A pointer The document handle
`out`	*pdf_page*	A pointer to a page handle
`in`	*addgrid*	Set to TRUE to add coordinate grid lines to the paper (in points)

16.1.3.6 stroke_to_file()

```
int stroke_to_file (
            HPDF_Doc pdf_doc,
            int argc,
            char ** argv )
```

Stroke the created PDF page to a file.

The filename to be used is determined by the setup_filename() function

Parameters

pdf_doc	The PDF document to be written
argc	The main() functions argc argument
argv	The main() functions argv argument

Returns

> 0 on success, -1 on failure

See also

> setup_filename(), hpdftbl_stroke_pdfdoc()

Examples

> example01.c.

16.1.4 Variable Documentation

16.1.4.1 _hpdftbl_jmp_env

```
jmp_buf _hpdftbl_jmp_env
```

For simulated exception handling.

This specifies the setjmp() buffer to be used in the error handling routine to jump (via a longjmp()) to a specfied recovery point.

See also

> table_error_handler()

Examples

> example01.c.

16.1.4.2 run_as_unit_test

```
_Bool run_as_unit_test = FALSE
```

For the case when we use this example as a unit/integration test we do not want data such as dates, times, and system-information to be updated since the checks compare the result to a stored copy of the PDF file.

When this is true it is used as a flag in, for example, the call-back functions to suppress the printing of dates and times so the result may be compared to the styored versions of the document and not be dependent on a specific time and date.

Examples

example01.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex14.c, and tut_ex30.c.

16.2 unit_test.inc.h

Go to the documentation of this file.
```
1
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <execinfo.h>
13
14 #if !(defined _WIN32 || defined __WIN32__)
15 #include <unistd.h>
16 #include <libgen.h>
17 #include <sys/stat.h>
18 #include <sys/utsname.h>
19 #endif
20 #include <hpdf.h>
21 #include <math.h>
22 #include <setjmp.h>
23 #include <time.h>
24 #include "../src/hpdftbl.h"
25
35 _Bool run_as_unit_test = FALSE;
36
45 jmp_buf _hpdftbl_jmp_env;
46
47 #ifndef _MSC_VER
48 // Silent gcc about unused "arg" in the callback and error functions
49 #pragma GCC diagnostic push
50 #pragma GCC diagnostic ignored "-Wunused-parameter"
51 #pragma GCC diagnostic ignored "-Wunused-function"
52 #endif
53
54 #define TESTS_DIR "./tests/"
55
62 static void
63 error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
64                       void *user_data) {
65     fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
66             hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
67     longjmp(_hpdftbl_jmp_env, 1);
68 }
69
82 static void
83 table_error_handler(hpdftbl_t t, int r, int c, int err) {
84
85     int lineno;
86     char *filename;
87     char *extrainfo;
88
89     hpdftbl_get_last_err_file(&lineno, &filename, &extrainfo);
90     if (r > -1 && c > -1) {
91         fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)", err,
92                 hpdftbl_get_errstr(err), r, c);
93     } else {
94         fprintf(stderr, "*** Table Error: [%d] \"%s\"", err,
95                 hpdftbl_get_errstr(err));
96     }
97     if( filename != NULL ) {
98         fprintf(stderr," in %s:%d",filename, lineno);
```

```
 99        }
100        if( extrainfo != NULL ) {
101            fprintf(stderr,". Info: \"%s\"\n",extrainfo);
102        }
103        else {
104            fprintf(stderr,"\n");
105        }
106
107        // Also print the available stacktrace
108        void* callstack[128];
109        int i, frames = backtrace(callstack, 128);
110        char** callstack_sym = backtrace_symbols(callstack, frames);
111        if( callstack_sym != NULL ) {
112            fprintf(stderr, "Stacktrace:\n");
113            for (i = 0; i < frames; ++i) {
114                fprintf(stderr, "%s\n", callstack_sym[i]);
115            }
116            free(callstack_sym);
117        }
118
119        longjmp(_hpdftbl_jmp_env, 1);
120 }
121
122
123 #ifndef _MSC_VER
124 #pragma GCC diagnostic pop
125 #endif
126
127 // Setup a new PDF document with one page
150 void
151 setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
152     *pdf_doc = HPDF_New(error_handler, NULL);
153     *pdf_page = HPDF_AddPage(*pdf_doc);
154     HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
155     HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
156     if (addgrid) {
157         hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
158     }
159 }
160
180 char *
181 setup_filename(int argc, char **argv) {
182     static char file[1024];
183     if ( 2==argc ) {
184         strncpy(file, argv[1], sizeof file);
185         file[sizeof(file)-1] = 0;
186     } else if ( 1==argc ) {
187         char fbuff[255];
188         strncpy(fbuff, argv[0], sizeof fbuff);
189         fbuff[sizeof(fbuff) - 1] = 0;
190         char *bname = basename(fbuff);
191         snprintf(file, sizeof file, "out/%s.pdf", bname);
192     } else {
193         return NULL;
194     }
195     return file;
196 }
197
209 int
210 stroke_to_file(HPDF_Doc pdf_doc, int argc, char **argv) {
211     char *file;
212     if( NULL == (file=setup_filename(argc, argv)) ) {
213         fprintf(stderr,"ERROR: Unknown arguments!\n");
214         return -1;
215     }
216
217     printf("Sending to file \"%s\" ...\n", file);
218     if ( -1 == hpdftbl_stroke_pdfdoc(pdf_doc, file) ) {
219         fprintf(stderr,"ERROR: Cannot save to file. Does the full directory path exist?\n");
220         return -1;
221     }
222     printf("Done.\n");
223     return 0;
224 }
225
226
230 typedef char **content_t;
231
238 void setup_dummy_content(content_t *content, size_t rows, size_t cols) {
239     char buff[255];
240     *content = calloc(rows*cols, sizeof(char*));
241     size_t cnt = 0;
242     for (size_t r = 0; r < rows; r++) {
243         for (size_t c = 0; c < cols; c++) {
244             snprintf(buff, sizeof(buff), "Content %zu", cnt);
245             (*content)[cnt] = strdup(buff);
246             cnt++;
```

```
247            }
248        }
249 }
250
258 void setup_dummy_content_label(content_t *content, content_t *labels, size_t rows, size_t cols) {
259     char buff[255];
260     *content = calloc(rows*cols, sizeof(char*));
261     *labels = calloc(rows*cols, sizeof(char*));
262     size_t cnt = 0;
263     for (size_t r = 0; r < rows; r++) {
264         for (size_t c = 0; c < cols; c++) {
265             snprintf(buff, sizeof(buff), "Content %zu", cnt);
266             (*content)[cnt] = strdup(buff);
267             snprintf(buff, sizeof(buff), "Label %zu:", cnt);
268             (*labels)[cnt] = strdup(buff);
269             cnt++;
270         }
271     }
272 }
273
294 char *
295 mkfullpath(char *filename) {
296     const size_t len=strlen(filename)+strlen(TESTS_DIR)+1;
297     char *fullpath = calloc(len, sizeof(char));
298     if( NULL==fullpath ) {
299         fprintf(stderr,"Error: Failed to allocate dynamic buffer (%s:%d)\n", __FUNCTION__ , __LINE__);
300         exit(1);
301     }
302     strlcat(fullpath,TESTS_DIR,len);
303     strlcat(fullpath,filename,len);
304     return fullpath;
305 }
306
307
315 #define TUTEX_MAIN(_tbl_, _showgrid_) int \
316 main(int argc, char **argv) { \
317     HPDF_Doc pdf_doc; \
318     HPDF_Page pdf_page;    \
319     run_as_unit_test = 2==argc ; \
320     if (setjmp(_hpdftbl_jmp_env)) { \
321         return EXIT_FAILURE; \
322     }  \
323     hpdftbl_set_errhandler(table_error_handler); \
324     setup_hpdf(&pdf_doc, &pdf_page, _showgrid_); \
325     _tbl_(pdf_doc, pdf_page); \
326     if( -1 == stroke_to_file(pdf_doc, argc, argv) ) \
327         return EXIT_FAILURE; \
328     else \
329         return EXIT_SUCCESS; \
330 }
331
```

16.3 bootstrap.sh File Reference

Bootstrap the autotools environment and configure a build setup.

Functions

- h usage **$** (basename $0)" exit 0

Variables

- set u declare s **ORIG_DIR**

    The original directory from where this script is run.
- do if getopts ch **option**
- then case $option in c really_clean **exit**
- q **quiet_flag** =1

16.3.1 Detailed Description

Bootstrap the autotools environment and configure a build setup.

Note

> This must be run when the source have been obtained by cloning the repo and requires a full installation of GNU autotools as a pre-requisite.

Usage:

bootstrap.sh [-q] [-h]

-c : Clean **all** generated files. This is equivalent with cloning from the repo.
-q : Quiet
-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson `johan162@gmail.com`

16.4 dbgbld.sh File Reference

Setup a build environment for debugging.

Functions

- then case $option in h usage **$** (basename $0)" exit 0

Variables

- declare r **ORIG_DIR**

    The original directory from where this script is run.
- do if getopts ch **option**
- q **quiet_flag** =1

16.4.1 Detailed Description

Setup a build environment for debugging.

In order for easy debugging this means that the debug configuration will only build static library in order to be able to include it in the binaries (e.g. the example programs). With dynamic libraries not yet installed the libtools will build wrapper shell scripts which cannot be debugged.

Usage:

dbgbld.sh [-q] [-h]

-q : Quiet
-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson `johan162@gmail.com`

16.5 stdbld.sh File Reference

Setup a build environment for production build.

Functions

- then case $option in h usage **$** (basename $0)" exit 0

Variables

- declare r **ORIG_DIR**

    The original directory from where this script is run.
- do if getopts ch **option**
- q **quiet_flag** =1

16.5.1 Detailed Description

Setup a build environment for production build.

Usage:

stdbld.sh [-q] [-h]

-q : Quiet
-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson   johan162@gmail.com

16.6 config.h

```
1 /* src/config.h.  Generated from config.h.in by configure.  */
2 /* src/config.h.in.  Generated from configure.ac by autoheader.  */
3
4 /* Define to 1 if you have the <dlfcn.h> header file. */
5 #define HAVE_DLFCN_H 1
6
7 /* Define to 1 if you have the <hpdf.h> header file. */
8 #define HAVE_HPDF_H 1
9
10 /* Define to 1 if you have the <inttypes.h> header file. */
11 #define HAVE_INTTYPES_H 1
12
13 /* Define to 1 if you have the 'hpdf' library (-lhpdf). */
14 #define HAVE_LIBHPDF 1
15
16 /* Define to 1 if you have the 'iconv' library (-liconv). */
17 #define HAVE_LIBICONV 1
18
19 /* True if system have json library jansson */
20 #define HAVE_LIBJANSSON 1
21
22 /* Define to 1 if you have the <stdint.h> header file. */
23 #define HAVE_STDINT_H 1
24
25 /* Define to 1 if you have the <stdio.h> header file. */
26 #define HAVE_STDIO_H 1
27
28 /* Define to 1 if you have the <stdlib.h> header file. */
29 #define HAVE_STDLIB_H 1
30
31 /* Define to 1 if you have the <strings.h> header file. */
32 #define HAVE_STRINGS_H 1
```

```
33
34 /* Define to 1 if you have the <string.h> header file. */
35 #define HAVE_STRING_H 1
36
37 /* Define to 1 if you have the <sys/stat.h> header file. */
38 #define HAVE_SYS_STAT_H 1
39
40 /* Define to 1 if you have the <sys/types.h> header file. */
41 #define HAVE_SYS_TYPES_H 1
42
43 /* Define to 1 if you have the <unistd.h> header file. */
44 #define HAVE_UNISTD_H 1
45
46 /* True if system type is Linux */
47 #define IS_LINUX 0
48
49 /* True if system type is Apple OSX */
50 #define IS_OSX 1
51
52 /* Define to the sub-directory where libtool stores uninstalled libraries. */
53 #define LT_OBJDIR ".libs/"
54
55 /* Name of package */
56 #define PACKAGE "libhpdftbl"
57
58 /* Define to the address where bug reports for this package should be sent. */
59 #define PACKAGE_BUGREPORT "johan162@gmail.com"
60
61 /* Define to the full name of this package. */
62 #define PACKAGE_NAME "libhpdftbl"
63
64 /* Define to the full name and version of this package. */
65 #define PACKAGE_STRING "libhpdftbl 1.4.0"
66
67 /* Define to the one symbol short name of this package. */
68 #define PACKAGE_TARNAME "libhpdftbl"
69
70 /* Define to the home page for this package. */
71 #define PACKAGE_URL ""
72
73 /* Define to the version of this package. */
74 #define PACKAGE_VERSION "1.4.0"
75
76 /* Define to 1 if all of the C90 standard headers exist (not just the ones
77    required in a freestanding environment). This macro is provided for
78    backward compatibility; new code need not use it. */
79 #define STDC_HEADERS 1
80
81 /* Version number of package */
82 #define VERSION "1.4.0"
```

16.7 hpdftbl.c File Reference

Main module for flexible table drawing with HPDF library.

```
#include <stdlib.h>
#include <string.h>
#include <iconv.h>
#include <hpdf.h>
#include <libgen.h>
#include <sys/stat.h>
#include "hpdftbl.h"
```

Data Structures

- struct line_dash_style

    Definition of a dashed line style.

Typedefs

- typedef struct line_dash_style **line_dash_style_t**

    Definition of a dashed line style.

Functions

- int hpdftbl_set_line_dash (hpdftbl_t t, hpdftbl_line_dashstyle_t style)

    Internal helper to set the line style.
- void hpdftbl_set_anchor_top_left (hpdftbl_t tbl, const _Bool anchor)

    Switch stroking anchor point.
- _Bool hpdftbl_get_anchor_top_left (hpdftbl_t tbl)

    Get stroking anchor point.
- void hpdftbl_set_text_encoding (char ∗target, char ∗source)

    Determine text source encoding.
- int hpdftbl_encoding_text_out (HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, char ∗text)

    Strke text with current encoding.
- void HPDF_RoundedCornerRectangle (HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF↩
    _REAL width, HPDF_REAL height, HPDF_REAL rad)

    Draw rectangle with rounded corner.
- void hpdftbl_set_bottom_vmargin_factor (hpdftbl_t t, HPDF_REAL f)

    The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:
- hpdftbl_t hpdftbl_create (size_t rows, size_t cols)

    Create a new table with no title.
- hpdftbl_t hpdftbl_create_title (size_t rows, size_t cols, char ∗title)

    Create a new table with title top row.
- int hpdftbl_set_min_rowheight (hpdftbl_t t, float h)

    Set the minimum row height in the table.
- int hpdftbl_set_colwidth_percent (hpdftbl_t t, size_t c, float w)

    Set column width as percentage of overall table width.
- int hpdftbl_set_outer_grid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

    Set outer border grid style.
- int hpdftbl_set_inner_grid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

    Set inner border grid style.
- int hpdftbl_set_inner_hgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

    Set inner horizontal border grid style.
- int hpdftbl_set_inner_vgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

    Set inner vertical border grid style.
- int hpdftbl_set_inner_tgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

    Set inner horizontal top border grid style.
- int hpdftbl_set_zebra (hpdftbl_t t, _Bool use, int phase)
- int hpdftbl_set_zebra_color (hpdftbl_t t, HPDF_RGBColor z1, HPDF_RGBColor z2)

    Specify first and second color for a zebra grid table.
- int hpdftbl_set_header_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩
    RGBColor background)

Specify style for table header row.

- int hpdftbl_set_background (hpdftbl_t t, HPDF_RGBColor background)

  Set table background color.

- int hpdftbl_set_header_halign (hpdftbl_t t, hpdftbl_text_align_t align)

  Set table header horizontal text align.

- int hpdftbl_use_header (hpdftbl_t t, _Bool use)

  Enable/disable the interpretation of the top row as a header row.

- int hpdftbl_use_labels (hpdftbl_t t, _Bool use)

  Enable/Disable the use of cell labels.

- int hpdftbl_use_labelgrid (hpdftbl_t t, _Bool use)

  Shorter vertical line to mark labels.

- int hpdftbl_set_tag (hpdftbl_t t, void ∗tag)

  Set an optional tag for the table.

- int hpdftbl_destroy (hpdftbl_t t)

  Destroy a table and free all memory.

- _Bool chktbl (hpdftbl_t t, size_t r, size_t c)

  Internal function. Check that a row and column are within the table.

- int hpdftbl_set_cell (hpdftbl_t t, size_t r, size_t c, char ∗label, char ∗content)

  Set content for specific cell.

- int hpdftbl_set_cellspan (hpdftbl_t t, size_t r, size_t c, size_t rowspan, size_t colspan)

  Set cell spanning.

- int hpdftbl_clear_spanning (hpdftbl_t t)

  Clear all cell spanning.

- int hpdftbl_set_labels (hpdftbl_t t, char ∗∗labels)

  Set the text for the cell labels.

- int hpdftbl_set_content (hpdftbl_t t, char ∗∗content)

  Set the content for the table.

- int hpdftbl_set_label_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set the text style for labels in the entire table.

- int hpdftbl_set_content_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set text style for text content.

- int hpdftbl_set_row_content_style (hpdftbl_t t, size_t r, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor background)

  Set the text style for an entire row of cells.

- int hpdftbl_set_col_content_style (hpdftbl_t t, size_t c, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor background)

  Set the text style for an entire column of cells.

- int hpdftbl_set_cell_content_style (hpdftbl_t t, size_t r, size_t c, char ∗font, HPDF_REAL fsize, HPDF_↩ RGBColor color, HPDF_RGBColor background)

  Set the text style for content of specified cell.

- int hpdftbl_set_title_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set the table title text style.

- int hpdftbl_set_title (hpdftbl_t t, char ∗title)

  Set table title.

- int hpdftbl_set_title_halign (hpdftbl_t t, hpdftbl_text_align_t align)

  Set horizontal alignment for table title.

- int hpdftbl_stroke_from_data (HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t ∗tbl_spec, hpdftbl_theme_t ∗theme)

  Construct the table from a array specification.

- int hpdftbl_get_last_auto_height (HPDF_REAL ∗height)

  Get the height calculated for the last constructed table.
- int hpdftbl_setpos (hpdftbl_t t, const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, HPDF_REAL height)

  Set size and position for table.
- int hpdftbl_stroke_pos (HPDF_Doc pdf, const HPDF_Page page, hpdftbl_t t)

  Stroke the table using the already specified size and position within the table.
- int hpdftbl_stroke (HPDF_Doc pdf, const HPDF_Page page, hpdftbl_t t, const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, HPDF_REAL height)

  Stroke the table.
- int hpdftbl_stroke_pdfdoc (HPDF_Doc pdf_doc, char ∗file)

  Stroke PDF document to file with check that the directory in path exists.

Variables

- hpdftbl_error_handler_t **hpdftbl_err_handler** = NULL

  This stores a pointer to the function acting as the error handler callback.

16.7.1 Detailed Description

Main module for flexible table drawing with HPDF library.

Author

Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

LICENSE

Released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN-CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.7.2 Function Documentation

16.7.2.1 chktbl()

```
_Bool chktbl (
            hpdftbl_t t,
            size_t r,
            size_t c )
```

Internal function. Check that a row and column are within the table.

Internal function. Check that a row and column are within the table

Parameters

t	Table handle
r	Row
c	Column

Returns

TRUE if within bounds, FALSE otherwise

Referenced by hpdftbl_set_cell(), hpdftbl_set_cell_canvas_cb(), hpdftbl_set_cell_content_cb(), hpdftbl_set_cell_content_style(), hpdftbl_set_cell_content_style_cb(), hpdftbl_set_cell_label_cb(), and hpdftbl_set_cellspan().

16.7.2.2 HPDF_RoundedCornerRectangle()

```
void HPDF_RoundedCornerRectangle (
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            HPDF_REAL width,
            HPDF_REAL height,
            HPDF_REAL rad )
```

Draw rectangle with rounded corner.

Draw a rectangle with rounded corner with the current line width, color. The rectangle will not be stroked.

Parameters

page	Page handle
xpos	Lower left x-position of rectangle
ypos	Lower left y-position of rectangle
width	Width of rectangle
height	Height of rectangle
rad	Radius of corners

Referenced by [hpdftbl_widget_slide_button()](#).

16.7.2.3   hpdftbl_clear_spanning()

```
int hpdftbl_clear_spanning (
            hpdftbl_t t )
```

Clear all cell spanning.

Reset all spanning cells to no spanning

Parameters

t	Table handle

Returns

>   0 on success, -1 on failure

See also

>   [hpdftbl_set_cellspan()](#)

16.7.2.4   hpdftbl_create()

```
hpdftbl_t hpdftbl_create (
            size_t rows,
            size_t cols )
```

Create a new table with no title.

Create a new table structure. This is the basic handler needed for most other API functions.

Parameters

rows	Number of rows
cols	Number of columns

Returns

>   A handle to a table, NULL in case of OOM

Examples

>   [tut_ex01.c](#), [tut_ex02.c](#), [tut_ex02_1.c](#), [tut_ex03.c](#), [tut_ex04.c](#), [tut_ex09.c](#), [tut_ex10.c](#), [tut_ex11.c](#), [tut_ex12.c](#), [tut_ex15.c](#), [tut_ex15_1.c](#), [tut_ex20.c](#), [tut_ex40.c](#), and [tut_ex41.c](#).

16.7.2.5 hpdftbl_create_title()

```
hpdftbl_t hpdftbl_create_title (
            size_t rows,
            size_t cols,
            char * title )
```

Create a new table with title top row.

Create a new table structure. This is the basic handler needed for most other API functions.

Parameters

rows	Number of rows
cols	Number of columns
title	Title of table

Returns

> A handle to a table, NULL in case of OOM

Examples

> example01.c, tut_ex00.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex14.c, and tut_ex30.c.

Referenced by hpdftbl_create(), and hpdftbl_stroke_from_data().

16.7.2.6 hpdftbl_destroy()

```
int hpdftbl_destroy (
            hpdftbl_t t )
```

Destroy a table and free all memory.

Destroy a table previous created with table_create(), It is the calling routines responsibility not to acceess t again.

Parameters

t	Handle to table

Returns

> 0 on success, -1 on failure

Referenced by hpdftbl_stroke_from_data().

16.7.2.7 hpdftbl_encoding_text_out()

```
int hpdftbl_encoding_text_out (
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            char * text )
```

Strke text with current encoding.

Utility function to stroke text with character encoding. It is the calling routines responsibility to enclose text in a HPDF_Page_BeginText() / HPDF_Page_EndText()

Parameters

page	Page handle
xpos	X coordinate
ypos	Y coordinate
text	Text to print

Returns

-1 on error, 0 on success

16.7.2.8 hpdftbl_get_anchor_top_left()

```
_Bool hpdftbl_get_anchor_top_left (
            hpdftbl_t tbl )
```

Get stroking anchor point.

Get anchor point for table positioning. By default the top left is used.

Parameters

tbl	Table handle

See also

hpdftbl_set_anchor_top_left

Returns

TRUE if anchor is top left, FALSE otherwise

16.7.2.9 hpdftbl_get_last_auto_height()

```
int hpdftbl_get_last_auto_height (
            HPDF_REAL * height )
```

Get the height calculated for the last constructed table.

Get the last automatically calculated height when stroking a table. (The height will be automatically calculated if it was specified as 0)

Parameters

height	Returned height

Returns

-1 on error, 0 if successful

16.7.2.10 hpdftbl_set_anchor_top_left()

```
void hpdftbl_set_anchor_top_left (
            hpdftbl_t tbl,
            const _Bool anchor )
```

Switch stroking anchor point.

Set anchor point for table positioning. By default the top left is used as anchor. Calling this function with FALSE can sets the anchor to bottom left instead.

Parameters

tbl	Table handle
anchor	Set to TRUE to use top left as anchor, FALSE for bottom left

16.7.2.11 hpdftbl_set_background()

```
int hpdftbl_set_background (
            hpdftbl_t t,
            HPDF_RGBColor background )
```

Set table background color.

Parameters

t	Table handle
background	Background color

Returns

> 0 on success, -1 on failure

16.7.2.12 hpdftbl_set_bottom_vmargin_factor()

```
void hpdftbl_set_bottom_vmargin_factor (
            hpdftbl_t t,
            HPDF_REAL f )
```

The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:

```
bottom_margin = fontsize * f
```

The default margin is specified by the define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR

Parameters

t	Table handle
f	Bottom margin factor

Referenced by hpdftbl_apply_theme().

16.7.2.13 hpdftbl_set_cell()

```
int hpdftbl_set_cell (
            hpdftbl_t t,
            size_t r,
            size_t c,
            char * label,
            char * content )
```

Set content for specific cell.

Set label and content for a specific cell. If the specified cell is part of another cells spanning an error occurs (returns -1),

Parameters

t	Table handle
r	Row
c	Column
label	Label
content	Text content

Returns

-1 on error, 0 if successful

Examples

tut_ex01.c, and tut_ex03.c.

Referenced by hpdftbl_stroke_from_data().

16.7.2.14 hpdftbl_set_cell_content_style()

```
int hpdftbl_set_cell_content_style (
            hpdftbl_t t,
            size_t r,
            size_t c,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for content of specified cell.

SSet the font style for content of specified cell. This will override the global cell content setting.

Parameters

t	Table handle
r	Cell row
c	Cell column
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

0 on success, -1 on failure

See also

hpdftbl_set_content_style()

hpdftbl_set_cell_content_style_cb()

Examples

example01.c.

Referenced by hpdftbl_set_col_content_style(), and hpdftbl_set_row_content_style().

16.7.2.15 hpdftbl_set_cellspan()

```
int hpdftbl_set_cellspan (
            hpdftbl_t t,
            size_t r,
            size_t c,
            size_t rowspan,
            size_t colspan )
```

Set cell spanning.

Set row and column spanning for a cell, an expanded cell is referenced via the position of it's top-left cell

Parameters

t	Table handle
r	Row
c	Column
rowspan	Row span
colspan	Column span

Returns

-1 on error, 0 if successful

See also

hpdftbl_clear_spanning()

Examples

example01.c, tut_ex07.c, tut_ex08.c, tut_ex40.c, and tut_ex41.c.

Referenced by hpdftbl_stroke_from_data().

16.7.2.16 hpdftbl_set_col_content_style()

```
int hpdftbl_set_col_content_style (
            hpdftbl_t t,
            size_t c,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for an entire column of cells.

Set font options for the specified column of cells. This will override the global cell content setting.

Parameters

t	Table handle
c	Column to affect
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

0 on success, -1 on failure

See also

hpdftbl_set_content_style()

hpdftbl_set_cell_content_style_cb()

16.7.2.17 hpdftbl_set_colwidth_percent()

```
int hpdftbl_set_colwidth_percent (
            hpdftbl_t t,
            size_t c,
            float w )
```

Set column width as percentage of overall table width.

Specify column width as percentage of total column width. Note that this will only take effect if the table has an overall width specified when stroked. Too avoid errors one column should be left unspecified to let the library use whatever space is left for that column.

Parameters

t	Table handle
c	Column to set width of first column has index 0
w	Width as percentage in range [0.0, 100.0]

Returns

0 on success, -1 on failure

Examples

example01.c, tut_ex08.c, tut_ex09.c, tut_ex10.c, tut_ex11.c, and tut_ex12.c.

16.7.2.18 hpdftbl_set_content()

```
int hpdftbl_set_content (
            hpdftbl_t t,
            char ** content )
```

Set the content for the table.

Set content for all cells. It is the calling functions responsibility that the content array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r ∗ num_cols + c) where num_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with strdup() so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N∗M) entries.

Another way to specify the content is to use the callback mechanism. By setting up a content callback function that returns the content for a cell.

Parameters

t	Table handle
content	A one dimensional string array of content string

Returns

-1 on error, 0 if successful

See also

hpdftbl_set_content_callback()

hpdftbl_set_cell_content_callback()

Examples

example01.c, tut_ex00.c, tut_ex02.c, tut_ex02_1.c, tut_ex04.c, tut_ex05.c, tut_ex10.c, tut_ex11.c, tut_ex12.c, tut_ex15.c, tut_ex15_1.c, tut_ex20.c, tut_ex40.c, and tut_ex41.c.

16.7.2.19 hpdftbl_set_content_style()

```
int hpdftbl_set_content_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set text style for text content.

Set style options for cell content (font, color, background). This will be applied for all cells in the table. If a style callback have been specified for either the table or a cell that style take precedence.

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

-1 on error, 0 if successful

See also

hpdftbl_set_cell_content_style()

hpdftbl_set_cell_content_style_cb()

Examples

example01.c.

Referenced by hpdftbl_apply_theme().

16.7.2.20 hpdftbl_set_header_halign()

```
int hpdftbl_set_header_halign (
            hpdftbl_t t,
            hpdftbl_text_align_t align )
```

Set table header horizontal text align.

Parameters

t	Table handle
align	Alignment

Returns

0 on success, -1 on failure

Referenced by hpdftbl_apply_theme().

16.7.2.21 hpdftbl_set_header_style()

```
int hpdftbl_set_header_style (
            hpdftbl_t t,
```

```
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Specify style for table header row.

Set the font properties and background for the header row which is the top row if enabled. The header row will be automatically enabled after calling this function. The header can be enabled/disabled separately with hpdftbl_use_header()

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Font color
background	Cell background color

Returns

0 on success, -1 on failure

See also

hpdftbl_use_header()

Referenced by hpdftbl_apply_theme().

16.7.2.22 hpdftbl_set_inner_grid_style()

```
int hpdftbl_set_inner_grid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set inner border grid style.

This is a shortform to set both the vertical and horizontal gridline style with one call.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

0 on success, -1 on failure

See also

hpdftbl_set_inner_hgrid_style(), hpdftbl_set_inner_vgrid_style(), hpdftbl_set_outer_grid_style()

16.7.2.23 hpdftbl_set_inner_hgrid_style()

```
int hpdftbl_set_inner_hgrid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set inner horizontal border grid style.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

0 on success, -1 on failure

See also

hpdftbl_set_inner_grid_style(), hpdftbl_set_inner_vgrid_style()

Examples

tut_ex15_1.c, and tut_ex20.c.

Referenced by hpdftbl_apply_theme(), and hpdftbl_set_inner_grid_style().

16.7.2.24 hpdftbl_set_inner_tgrid_style()

```
int hpdftbl_set_inner_tgrid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set inner horizontal top border grid style.

This would be the gridline just below the header row.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

0 on success, -1 on failure

See also

hpdftbl_set_inner_hgrid_style()

Examples

tut_ex15_1.c, and tut_ex20.c.

Referenced by hpdftbl_apply_theme().

16.7.2.25 hpdftbl_set_inner_vgrid_style()

```
int hpdftbl_set_inner_vgrid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set inner vertical border grid style.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

0 on success, -1 on failure

See also

hpdftbl_set_inner_grid_style(), hpdftbl_set_inner_hgrid_style()

Examples

tut_ex20.c.

Referenced by hpdftbl_apply_theme(), and hpdftbl_set_inner_grid_style().

16.7.2.26 hpdftbl_set_label_style()

```
int hpdftbl_set_label_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for labels in the entire table.

Set font, color and background options for cell labels. If a style callback have been specified for either the table or a cell that style take precedence.

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

-1 on error, 0 if successful

Referenced by hpdftbl_apply_theme().

16.7.2.27 hpdftbl_set_labels()

```
int hpdftbl_set_labels (
            hpdftbl_t t,
            char ** labels )
```

Set the text for the cell labels.

Set labels for all the cell. It is the calling functions responsibility that the labels array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r ∗ num_cols + c) where num_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with strdup() so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N∗M) entries.

Parameters

t	Table handle
labels	A one dimensional string array of labels

Returns

> -1 on error, 0 if successful

See also

> hpdftbl_set_cell_label_cb()
>
> hpdftbl_set_label_cb()

Examples

> example01.c, tut_ex00.c, tut_ex04.c, tut_ex05.c, tut_ex20.c, tut_ex40.c, and tut_ex41.c.

16.7.2.28   hpdftbl_set_line_dash()

```
int hpdftbl_set_line_dash (
            hpdftbl_t t,
            hpdftbl_line_dashstyle_t style )
```

Internal helper to set the line style.

The drawing of a dashed line uses the underlying HPDF function HPDF_Page_SetDash()

Parameters

t	Table handle
style	

Returns

> -1 on error, 0 on success

See also

> line_dash_style

16.7.2.29 hpdftbl_set_min_rowheight()

```
int hpdftbl_set_min_rowheight (
            hpdftbl_t t,
            float h )
```

Set the minimum row height in the table.

The row height is normally calculated based on the font size and if labels are displayed or not. However, it is not possible for the table to know the height of specific widgets (for example) without a two-pass table drawing algorithm.

To handle thos odd cases when the calculated height is not sufficient a manual minimum height can be specified.

Parameters

t	Table handler
h	The minimum height (in points). If specified as 0 the min height will have no effect.

Returns

> 0 on success, -1 on failure

Examples

> example01.c.

16.7.2.30 hpdftbl_set_outer_grid_style()

```
int hpdftbl_set_outer_grid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set outer border grid style.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

> 0 on success, -1 on failure

See also

[hpdftbl_set_inner_grid_style()](#)

Examples

[tut_ex20.c.](#)

Referenced by [hpdftbl_apply_theme()](#).

16.7.2.31 hpdftbl_set_row_content_style()

```
int hpdftbl_set_row_content_style (
            hpdftbl_t t,
            size_t r,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for an entire row of cells.

Set font options for the specified row of cells. This will override the global cell content.

Parameters

t	Table handle
r	Row to affect
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

0 on success, -1 on failure

See also

[hpdftbl_set_content_style()](#)

[hpdftbl_set_cell_content_style_cb()](#)

16.7.2.32 hpdftbl_set_tag()

```
int hpdftbl_set_tag (
            hpdftbl_t t,
            void * tag )
```

Set an optional tag for the table.

Set an optional tag in the table. The tag can be a pointer to anything. The tag is passed as the first argument in the various callbacks and can be used to supply table specific information or identify a specific table in the case the same callback is used for multiple tables.

Parameters

t	The table handle
tag	The tag (pointer to any object)

Returns

> 0 on success, -1 on failure

16.7.2.33 hpdftbl_set_text_encoding()

```
void hpdftbl_set_text_encoding (
            char * target,
            char * source )
```

Determine text source encoding.

The default HPDF encoding is a standard PDF encoding. The problem with that is that now almost 100% of all code is written in UTF-8 encoding and trying to print text strings with accented characters will simply not work. For example the default encoding assumes that strings are given in UTF-8 and sets the target to ISO8859-4 which includes northern europe accented characters. The conversion is internally handled by the standard iconv() routines.

Parameters

target	The target encoding. See HPDF documentation for supported encodings.
source	The source encodings, i.e. what encodings are sth strings in the source specified in.

16.7.2.34 hpdftbl_set_title()

```
int hpdftbl_set_title (
            hpdftbl_t t,
            char * title )
```

Set table title.

Set table title. A title will occupy a separate row above the table that is not included in the row count. A table is enabled when the table text is <> NULL and disabled when the title text is == NULL.

Parameters

t	Table handle
title	Title string

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_title_style()
>
> hpdftbl_set_title_halign()

16.7.2.35 hpdftbl_set_title_halign()

```
int hpdftbl_set_title_halign (
            hpdftbl_t t,
            hpdftbl_text_align_t align )
```

Set horizontal alignment for table title.

Parameters

t	Table handle
align	Alignment

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_title()
>
> hpdftbl_set_title_style()

Examples

> example01.c.

Referenced by hpdftbl_apply_theme().

16.7.2.36 hpdftbl_set_title_style()

```
int hpdftbl_set_title_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the table title text style.

Set font options for title

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

0 on success, -1 on failure

See also

hpdftbl_set_title()

hpdftbl_set_title_halign()

Examples

example01.c.

Referenced by hpdftbl_apply_theme().

16.7.2.37 hpdftbl_set_zebra()

```
int hpdftbl_set_zebra (
            hpdftbl_t t,
            _Bool use,
            int phase )
```

Parameters

t	Table handle
use	TRUE=Use Zebra, FALSE=Don't use zebra
phase	0=Start with color 1, 1=Start with color 1

Returns

0 on successes -1 on failure

Examples

tut_ex15.c, tut_ex15_1.c, and tut_ex41.c.

Referenced by hpdftbl_apply_theme().

16.7.2.38   hpdftbl_set_zebra_color()

```
int hpdftbl_set_zebra_color (
            hpdftbl_t t,
            HPDF_RGBColor z1,
            HPDF_RGBColor z2 )
```

Specify first and second color for a zebra grid table.

By default the colors start with z1 color. To have the top row (below any potential header row) instead start with z2 specify `phase=1` in the hpdftbl_set_zebra() function.

Parameters

t	Table handle
z1	Color 1
z2	Color 2

Returns

0 on successes -1 on failure

Referenced by hpdftbl_apply_theme().

16.7.2.39   hpdftbl_setpos()

```
int hpdftbl_setpos (
            hpdftbl_t t,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            HPDF_REAL height )
```

Set size and position for table.

The position is by default specified as the upper left corner of the table. Use the hpdftbl_set_origin_top_left() to use the bottom left of the table as reference point.

Ths standard stroke function hpdftbl_stroke() also take the size and position as argument for ease of use but the hpdftbl_stroke_pos() do assume that the table has it's size set.

Parameters

t	Table handle
xpos	x position for table
ypos	y position for table
width	width of table
height	height of table. If the height is specified as 0 it will be automatically calculated. The calculated height can be retrieved after the table has been stroked by a call to hpdftbl_get_last_auto_height()

Returns

-1 on error, 0 if successful

See also

hpdftbl_get_last_auto_height(), hpdftbl_set_origin_top_left()

16.7.2.40 hpdftbl_stroke()

```
int hpdftbl_stroke (
            HPDF_Doc pdf,
            const HPDF_Page page,
            hpdftbl_t t,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            HPDF_REAL height )
```

Stroke the table.

Stroke the table at the specified position and size. The position is by default specified as the upper left corner of the table. Use the hpdftbl_set_origin_top_left(FALSE) to use the bottom left of the table as reference point.

Parameters

pdf	The HPDF document handle
page	The HPDF page handle
t	Table handle
xpos	x position for table
ypos	y position for table
width	width of table
height	height of table. If the height is specified as 0 it will be automatically calculated. The calculated height can be retrieved after the table has been stroked by a call to hpdftbl_get_last_auto_height()

Returns

-1 on error, 0 if successful

See also

hpdftbl_get_last_auto_height()

hpdftbl_stroke_from_data()

Examples

example01.c, tut_ex00.c, tut_ex01.c, tut_ex02.c, tut_ex02_1.c, tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex09.c, tut_ex10.c, tut_ex11.c, tut_ex12.c, tut_ex14.c, tut_ex15.c, tut_ex15_1.c, tut_ex20.c, tut_ex30.c, tut_ex40.c, and tut_ex41.c.

Referenced by hpdftbl_stroke_from_data(), and hpdftbl_stroke_pos().

16.7.2.41 hpdftbl_stroke_from_data()

```
int hpdftbl_stroke_from_data (
            HPDF_Doc pdf_doc,
            HPDF_Page pdf_page,
            hpdftbl_spec_t * tbl_spec,
            hpdftbl_theme_t * theme )
```

Construct the table from a array specification.

Create and stroke a table specified by a data structure. This makes it easier to separate the view of the data from the model which provides the data. The intended use case is that the data structure specifies the core layout of the table together with the labels and callback functions to handle the content in each cell. Using this method to create a table also makes it much more maintainable.

Parameters

pdf_doc	The PDF overall document
pdf_page	The pageto stroke to
tbl_spec	The table specification
theme	Table theme to be applied

Returns

0 on success, -1 on failure

See also

hpdftbl_stroke()

Examples

example01.c, tut_ex13_1.c, and tut_ex13_2.c.

16.7.2.42 hpdftbl_stroke_pdfdoc()

```
int hpdftbl_stroke_pdfdoc (
            HPDF_Doc pdf_doc,
            char * file )
```

Stroke PDF document to file with check that the directory in path exists.

Note: It is a checked error if the full path is longer than 1014 characters

Parameters

pdf_doc	Haru PDF document handle
file	Full pathname of file to write to

Returns

> 0 on success, -1 on failure

Referenced by stroke_to_file().

16.7.2.43 hpdftbl_stroke_pos()

```
int hpdftbl_stroke_pos (
            HPDF_Doc pdf,
            const HPDF_Page page,
            hpdftbl_t t )
```

Stroke the table using the already specified size and position within the table.

Stroke the table at the specified position and size. The position is by default specified as the upper left corner of the table. Use the hpdftbl_set_origin_top_left(FALSE) to use the bottom left of the table as reference point.

This is a convenient method to use when stroking a serialized table as the table already holds the size and position. Stroking a table read back ccan be done with just two lines of code

```
hpdftbl_t tbl = calloc(1, sizeof(struct hpdftbl));
if( 0 == hpdftbl_load(tbl, filename)  ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

Parameters

pdf	The HPDF document handle
page	The HPDF page handle
t	Table handle

Returns

> -1 on error, 0 if successful

See also

> hpdftbl_get_last_auto_height()
>
> hpdftbl_setpos(), hpdftbl_stroke()

Examples

> tut_ex40.c, and tut_ex41.c.

16.7.2.44 hpdftbl_use_header()

```
int hpdftbl_use_header (
            hpdftbl_t t,
            _Bool use )
```

Enable/disable the interpretation of the top row as a header row.

A header row will have a different style and labels will be disabled on this row. In addition the text will be centered vertically and horizontal in the cell.

Parameters

t	Table handle
use	TRUE to enable, FALSE to disable

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_header_style()

Examples

> example01.c, tut_ex02_1.c, tut_ex11.c, tut_ex12.c, and tut_ex20.c.

Referenced by hpdftbl_stroke_from_data().

16.7.2.45   hpdftbl_use_labelgrid()

```
int hpdftbl_use_labelgrid (
            hpdftbl_t t,
            _Bool use )
```

Shorter vertical line to mark labels.

Set the usage of special grid style where the vertical grid only covers the label text and a gap to the next line. Horizontal lines are drawn as usual. The label grid style gives the table a "lighter" look.

Parameters

t	Table handle
use	TRUE to use label grid, FALSE o disable it

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_use_labels()

Examples

> example01.c, tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex14.c, tut_ex20.c, tut_ex30.c, and tut_ex40.c.

Referenced by hpdftbl_stroke_from_data().

16.7.2.46 hpdftbl_use_labels()

```
int hpdftbl_use_labels (
             hpdftbl_t t,
             _Bool use )
```

Enable/Disable the use of cell labels.

By default a newly created table will not use cell labels. Enabling labels will also by default enable the special label grid style. To adjust the grid style separately us the hpdftbl_use_labelgrid() method.

Parameters

t	Table handle
use	Set to TRUE for cell labels

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_use_labelgrid()

Examples

> example01.c, tut_ex00.c, tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, tut_ex14.c, tut_ex20.c, tut_ex30.c, tut_ex40.c, and tut_ex41.c.

Referenced by hpdftbl_stroke_from_data().

16.8 hpdftbl.h File Reference

Header file for libhpdftbl.

```
#include "config.h"
```

Data Structures

- struct text_style

    Specification of a text style.
- struct grid_style

    Specification for table grid lines.
- struct hpdftbl_cell

    Specification of individual cells in the table.
- struct hpdftbl

    Core table handle.
- struct hpdftbl_cell_spec

    Used in data driven table creation.
- struct hpdftbl_spec

    Used in data driven table creation.
- struct hpdftbl_theme

    Define a set of styles into a table theme.

Macros

- #define **hpdftbl_H**
- #define **TRUE** 1

    Boolean truth value.

- #define **FALSE** 0

    Boolean false value.

- #define **max**(a, b) (((a)>(b)) ? (a):(b))

    Return the maximum value of numeric variables.

- #define **min**(a, b) (((a)<(b)) ? (a):(b))

    Return the minimum value of numeric variables.

- #define THEME_JSON_VERSION 1
- #define TABLE_JSON_VERSION 1
- #define HPDF_FF_TIMES "Times-Roman"
- #define HPDF_FF_TIMES_ITALIC "Times-Italic"
- #define HPDF_FF_TIMES_BOLD "Times-Bold"
- #define HPDF_FF_TIMES_BOLDITALIC "Times-BoldItalic"
- #define HPDF_FF_HELVETICA "Helvetica"
- #define HPDF_FF_HELVETICA_ITALIC "Helvetica-Oblique"
- #define HPDF_FF_HELVETICA_BOLD "Helvetica-Bold"
- #define HPDF_FF_HELVETICA_BOLDITALIC "Helvetica-BoldOblique"
- #define HPDF_FF_COURIER "Courier"
- #define HPDF_FF_COURIER_BOLD "Courier-Bold"
- #define HPDF_FF_COURIER_IALIC "Courier-Oblique"
- #define HPDF_FF_COURIER_BOLDITALIC "Courier-BoldOblique"
- #define **HPDF_RGB_CONVERT**(r, g, b) (HPDF_RGBColor) { r / 255.0f, g / 255.0f, b / 255.0f }

    Utility macro to create a HPDF color constant from integer RGB values.

- #define **HPDF_COLOR_DARK_RED** (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
- #define **HPDF_COLOR_RED** (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
- #define **HPDF_COLOR_LIGHT_GREEN** (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
- #define **HPDF_COLOR_GREEN** (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
- #define **HPDF_COLOR_DARK_GREEN** (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
- #define **HPDF_COLOR_DARK_GRAY** (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
- #define **HPDF_COLOR_LIGHT_GRAY** (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
- #define **HPDF_COLOR_XLIGHT_GRAY** (HPDF_RGBColor) { 0.95f, 0.95f, 0.95f }
- #define **HPDF_COLOR_GRAY** (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
- #define **HPDF_COLOR_SILVER** (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
- #define **HPDF_COLOR_LIGHT_BLUE** (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
- #define **HPDF_COLOR_BLUE** (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
- #define **HPDF_COLOR_DARK_BLUE** (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
- #define **HPDF_COLOR_WHITE** (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
- #define **HPDF_COLOR_BLACK** (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
- #define **HPDF_COLOR_ORANGE** HPDF_RGB_CONVERT(0xF5, 0xD0, 0x98);
- #define **HPDF_COLOR_ALMOST_BLACK** HPDF_RGB_CONVERT(0x14, 0x14, 0x14);
- #define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5

    The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size.

- #define **HPDFTBL_DEFAULT_TARGET_ENCODING** "ISO8859-4"

    Default PDF text encodings.

- #define **HPDFTBL_DEFAULT_SOURCE_ENCODING** "UTF-8"

    Default input source text encodings.

- #define **A4PAGE_HEIGHT_CM** 29.7

    Standard A4 paper height in cm.

- #define **A4PAGE_WIDTH_CM** 21.0

Standard A4 paper width in cm.

- #define **A3PAGE_HEIGHT_CM** 42.0

  Standard A3 paper height in cm.

- #define **A3PAGE_WIDTH_CM** 29.7

  Standard A3 paper width in cm.

- #define **LETTERRPAGE_HEIGHT_CM** 27.9

  US Letter Height in cm.

- #define **LETTERRPAGE_WIDTH_CM** 21.6

  US Letter width in cm.

- #define **LEGALPAGE_HEIGHT_CM** 35.6

  US Legal Height in cm.

- #define **LEGALPAGE_WIDTH_CM** 21.6

  US Legal Width in cm.

- #define **HPDFTBL_END_CELLSPECS** {0, 0, 0, 0, 0, 0, 0, 0, 0}

  Sentinel to mark the end of Cell Specifications for data driven table definition.

- #define **HPDF_COLOR_FROMRGB**(r, g, b) (HPDF_RGBColor){(r)/255.0,(g)/255.0,(b)/255.0}

  Utility macro to calculate a color constant from RGB integer values [0,255].

- #define **HPDFTBL_MIN_CALCULATED_PERCENT_CELL_WIDTH** 2.0

  The smallest size in percent of table width allowed by automatic calculation before giving an error.

- #define hpdftbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54∗72)

  Convert cm to dots using the default resolution (72 DPI)

- #define _HPDFTBL_SET_ERR(t, err, r, c) do {hpdftbl_err_code=err;hpdftbl_err_row=r;hpdftbl_err_col=c;hpdftbl_err_lineno=↩
  __LINE__;hpdftbl_err_file=__FILE__; if(hpdftbl_err_handler){hpdftbl_err_handler(t,r,c,err);}} while(0)

  Call the error handler with specified error code and table row, col where error occured.

- #define _HPDFTBL_SET_ERR_EXTRA(info) do {strncpy(hpdftbl_err_extrainfo,info,1023);hpdftbl_err_extrainfo[1023]=0;}
  while(0)

  Set optional extra info at error state. (Currently only used by the late binding setting callback functions)

- #define **_HPDFTBL_CHK_TABLE**(t) do {if(NULL == t) {hpdftbl_err_code=-3;hpdftbl_err_row=-1;hpdftbl_err_col=-
  1;return -1;}} while(0)

  NPE check before using a table handler.

- #define **_HPDFTBL_IDX**(r, c) (r∗t->cols+c)

  Shortcut to calculate the index in an array from a row,column (table) position.

Typedefs

- typedef enum hpdftbl_text_align hpdftbl_text_align_t

  Enumeration for horizontal text alignment.

- typedef struct text_style hpdf_text_style_t

  Specification of a text style.

- typedef struct hpdftbl ∗ hpdftbl_t

  Table handle is a pointer to the hpdftbl structure.

- typedef char ∗(∗ hpdftbl_content_callback_t) (void ∗, size_t, size_t)

  Type specification for the table content callback.

- typedef void(∗ hpdftbl_canvas_callback_t) (HPDF_Doc, HPDF_Page, void ∗, size_t, size_t, HPDF_REAL,
  HPDF_REAL, HPDF_REAL, HPDF_REAL)

  Type specification for the table canvas callback.

- typedef _Bool(∗ hpdftbl_content_style_callback_t) (void ∗, size_t, size_t, char ∗content, hpdf_text_style_t ∗)

  Type specification for the content style.

- typedef void(∗ hpdftbl_callback_t) (hpdftbl_t)

  Callback type for optional post processing when constructing table from a data array.

- typedef enum hpdftbl_dashstyle hpdftbl_line_dashstyle_t

    Possible line dash styles for grid lines.
- typedef struct grid_style hpdftbl_grid_style_t

    Specification for table grid lines.
- typedef struct hpdftbl_cell hpdftbl_cell_t

    Type definition for the cell structure.
- typedef struct hpdftbl_cell_spec hpdftbl_cell_spec_t

    Used in data driven table creation.
- typedef struct hpdftbl_spec hpdftbl_spec_t

    Used in data driven table creation.
- typedef struct hpdftbl_theme hpdftbl_theme_t

    Define a set of styles into a table theme.
- typedef void(∗ hpdftbl_error_handler_t) (hpdftbl_t, int, int, int)

    TYpe for error handler function.

Enumerations

- enum hpdftbl_text_align { LEFT = 0 , CENTER = 1 , RIGHT = 2 }

    Enumeration for horizontal text alignment.
- enum hpdftbl_dashstyle {
    LINE_SOLID , LINE_DOT1 , LINE_DOT2 , LINE_DOT3 ,
    LINE_DOT4 , LINE_DASH1 , LINE_DASH2 , LINE_DASH3 ,
    LINE_DASH4 , LINE_DASH5 , LINE_DASHDOT1 , LINE_DASHDOT2 }

    Possible line dash styles for grid lines.

Functions

- hpdftbl_t hpdftbl_create (size_t rows, size_t cols)

    Create a new table with no title.
- hpdftbl_t hpdftbl_create_title (size_t rows, size_t cols, char ∗title)

    Create a new table with title top row.
- int hpdftbl_stroke (HPDF_Doc pdf, HPDF_Page page, hpdftbl_t t, HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height)

    Stroke the table.
- int hpdftbl_stroke_pos (HPDF_Doc pdf, const HPDF_Page page, hpdftbl_t t)

    Stroke the table using the already specified size and position within the table.
- int hpdftbl_stroke_from_data (HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t ∗tbl_spec, hpdftbl_theme_t ∗theme)

    Construct the table from a array specification.
- int hpdftbl_setpos (hpdftbl_t t, const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, HPDF_REAL height)

    Set size and position for table.
- int hpdftbl_destroy (hpdftbl_t t)

    Destroy a table and free all memory.
- int hpdftbl_get_last_auto_height (HPDF_REAL ∗height)

    Get the height calculated for the last constructed table.
- void hpdftbl_set_anchor_top_left (hpdftbl_t tbl, _Bool anchor)

    Switch stroking anchor point.
- _Bool hpdftbl_get_anchor_top_left (hpdftbl_t tbl)

    Get stroking anchor point.

- hpdftbl_error_handler_t hpdftbl_set_errhandler (hpdftbl_error_handler_t)

  Specify errhandler for the table routines.

- const char ∗ hpdftbl_get_errstr (int err)

  Translate a table error code to a human readable string.

- const char ∗ hpdftbl_hpdf_get_errstr (HPDF_STATUS err_code)

  Function to return a human readable error string for an error code from Core HPDF library.

- int hpdftbl_get_last_errcode (const char ∗∗errstr, int ∗row, int ∗col)

  Return last error code.

- void hpdftbl_get_last_err_file (int ∗lineno, char ∗∗file, char ∗∗extrainfo)

  Get the filename and line number where the last error occurred.

- void hpdftbl_default_table_error_handler (hpdftbl_t t, int r, int c, int err)

  A basic default table error handler.

- int hpdftbl_apply_theme (hpdftbl_t t, hpdftbl_theme_t ∗theme)

  Apply a specified theme to a table.

- hpdftbl_theme_t ∗ hpdftbl_get_default_theme (void)

  Return the default theme.

- int hpdftbl_get_theme (hpdftbl_t tbl, hpdftbl_theme_t ∗theme)

  Extract theme from settings of a specific table.

- int hpdftbl_destroy_theme (hpdftbl_theme_t ∗theme)

  Destroy existing theme structure and free memory.

- void hpdftbl_set_bottom_vmargin_factor (hpdftbl_t t, HPDF_REAL f)

  The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:

- int hpdftbl_set_min_rowheight (hpdftbl_t t, float h)

  Set the minimum row height in the table.

- int hpdftbl_set_colwidth_percent (hpdftbl_t t, size_t c, float w)

  Set column width as percentage of overall table width.

- int hpdftbl_clear_spanning (hpdftbl_t t)

  Clear all cell spanning.

- int hpdftbl_set_cellspan (hpdftbl_t t, size_t r, size_t c, size_t rowspan, size_t colspan)

  Set cell spanning.

- int hpdftbl_set_zebra (hpdftbl_t t, _Bool use, int phase)

- int hpdftbl_set_zebra_color (hpdftbl_t t, HPDF_RGBColor z1, HPDF_RGBColor z2)

  Specify first and second color for a zebra grid table.

- int hpdftbl_use_labels (hpdftbl_t t, _Bool use)

  Enable/Disable the use of cell labels.

- int hpdftbl_use_labelgrid (hpdftbl_t t, _Bool use)

  Shorter vertical line to mark labels.

- int hpdftbl_set_background (hpdftbl_t t, HPDF_RGBColor background)

  Set table background color.

- int hpdftbl_set_inner_tgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

  Set inner horizontal top border grid style.

- int hpdftbl_set_inner_vgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

  Set inner vertical border grid style.

- int hpdftbl_set_inner_hgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

  Set inner horizontal border grid style.

- int hpdftbl_set_inner_grid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

Set inner border grid style.

- int hpdftbl_set_outer_grid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)

  Set outer border grid style.
- int hpdftbl_set_header_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Specify style for table header row.
- int hpdftbl_set_header_halign (hpdftbl_t t, hpdftbl_text_align_t align)

  Set table header horizontal text align.
- int hpdftbl_use_header (hpdftbl_t t, _Bool use)

  Enable/disable the interpretation of the top row as a header row.
- int hpdftbl_set_label_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set the text style for labels in the entire table.
- int hpdftbl_set_row_content_style (hpdftbl_t t, size_t r, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor background)

  Set the text style for an entire row of cells.
- int hpdftbl_set_col_content_style (hpdftbl_t t, size_t c, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor background)

  Set the text style for an entire column of cells.
- int hpdftbl_set_content_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set text style for text content.
- int hpdftbl_set_cell_content_style (hpdftbl_t t, size_t r, size_t c, char ∗font, HPDF_REAL fsize, HPDF_↩ RGBColor color, HPDF_RGBColor background)

  Set the text style for content of specified cell.
- int hpdftbl_set_title_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set the table title text style.
- int hpdftbl_set_cell (hpdftbl_t t, size_t r, size_t c, char ∗label, char ∗content)

  Set content for specific cell.
- int hpdftbl_set_tag (hpdftbl_t t, void ∗tag)

  Set an optional tag for the table.
- int hpdftbl_set_title (hpdftbl_t t, char ∗title)

  Set table title.
- int hpdftbl_set_title_halign (hpdftbl_t t, hpdftbl_text_align_t align)

  Set horizontal alignment for table title.
- int hpdftbl_set_labels (hpdftbl_t t, char ∗∗labels)

  Set the text for the cell labels.
- int hpdftbl_set_content (hpdftbl_t t, char ∗∗content)

  Set the content for the table.
- int hpdftbl_set_content_cb (hpdftbl_t t, hpdftbl_content_callback_t cb)

  Set table content callback.
- int hpdftbl_set_cell_content_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb)

  Set cell content callback.
- int hpdftbl_set_label_cb (hpdftbl_t t, hpdftbl_content_callback_t cb)

  Set table label callback.
- int hpdftbl_set_cell_label_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb)

  Set cell label callback.
- int hpdftbl_set_canvas_cb (hpdftbl_t t, hpdftbl_canvas_callback_t cb)

  Set cell canvas callback.
- int hpdftbl_set_cell_canvas_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_canvas_callback_t cb)

Set cell canvas callback.
- int hpdftbl_set_content_style_cb (hpdftbl_t t, hpdftbl_content_style_callback_t cb)

  Set callback to specify cell content style.
- int hpdftbl_set_cell_content_style_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_content_style_callback_t cb)

  Set cell specific callback to specify cell content style.
- int hpdftbl_set_post_cb (hpdftbl_t t, hpdftbl_callback_t cb)

  Set table post processing callback.
- void hpdftbl_set_dlhandle (void ∗)

  Set the handle for scope of dynamic function search.
- int hpdftbl_set_content_dyncb (hpdftbl_t, const char ∗)

  Specify dynamic (late) loading callback content function.
- int hpdftbl_set_canvas_dyncb (hpdftbl_t, const char ∗)

  Specify dynamic (late) loading callback content function.
- int hpdftbl_set_cell_content_dyncb (hpdftbl_t, size_t, size_t, const char ∗)

  Specify dynamic (late) loading callback cell content function.
- int hpdftbl_set_label_dyncb (hpdftbl_t, const char ∗)

  Specify dynamic (late) loading callback for table label function.
- int hpdftbl_set_cell_label_dyncb (hpdftbl_t, size_t, size_t, const char ∗)

  Specify dynamic (late) loading callback for cell label function.
- int hpdftbl_set_content_style_dyncb (hpdftbl_t, const char ∗)

  Specify dynamic (late) loading callback for table style function.
- int hpdftbl_set_cell_content_style_dyncb (hpdftbl_t, size_t, size_t, const char ∗)

  Specify dynamic (late) loading callback for cell style function.
- int hpdftbl_set_cell_canvas_dyncb (hpdftbl_t, size_t, size_t, const char ∗)

  Specify dynamic (late) loading callback cell canvas function.
- int hpdftbl_set_post_dyncb (hpdftbl_t t, const char ∗cb_name)

  Set table post processing callback.
- void hpdftbl_set_text_encoding (char ∗target, char ∗source)

  Determine text source encoding.
- int hpdftbl_encoding_text_out (HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, char ∗text)

  Strke text with current encoding.
- void HPDF_RoundedCornerRectangle (HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF↩
  _REAL width, HPDF_REAL height, HPDF_REAL rad)

  Draw rectangle with rounded corner.
- void hpdftbl_stroke_grid (HPDF_Doc pdf, HPDF_Page page)
- void hpdftbl_table_widget_letter_buttons (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF_↩
  REAL ypos, HPDF_REAL width, HPDF_REAL height, HPDF_RGBColor on_color, HPDF_RGBColor off_↩
  color, HPDF_RGBColor on_background, HPDF_RGBColor off_background, HPDF_REAL fsize, const char
  ∗letters, _Bool ∗state)

  *Display an array of letters as a table where each letter is its own "mini" cell
  and sorrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different
  font and fac colors.*
- void hpdftbl_widget_slide_button (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF_REAL
  ypos, HPDF_REAL width, HPDF_REAL height, _Bool state)

  Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.
- void hpdftbl_widget_hbar (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos,
  HPDF_REAL width, HPDF_REAL height, HPDF_RGBColor color, float val, _Bool hide_val)

  Draw a horizontal partially filled bar to indicate an analog (percentage) value.
- void hpdftbl_widget_segment_hbar (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF_REAL
  ypos, HPDF_REAL width, HPDF_REAL height, size_t num_segments, HPDF_RGBColor on_color, double
  val_percent, _Bool hide_val)

  Draw a horizontal segment meter that can be used to visualize a discrete value.

- void hpdftbl_widget_strength_meter (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, size_t num_segments, HPDF_RGBColor on_color, size_t num_on_segments)

    Draw a phone strength meter.
- int hpdftbl_stroke_pdfdoc (HPDF_Doc pdf_doc, char ∗file)

    Stroke PDF document to file with check that the directory in path exists.
- int hpdftbl_dump (hpdftbl_t tbl, char ∗filename)

    Serialize a table structure as a JSON file.
- int hpdftbl_dumps (hpdftbl_t tbl, char ∗buff, size_t buffsize)

    Serialize a table structure to a string buffer.
- int hpdftbl_load (hpdftbl_t tbl, char ∗filename)

    Import a table structure from a serialized table on file.
- int hpdftbl_loads (hpdftbl_t tbl, char ∗buff)

    Import a table structure from a serialized json buffert.
- int hpdftbl_theme_dump (hpdftbl_theme_t ∗theme, char ∗filename)

    Serialize the specified theme structure to a named file.
- int hpdftbl_theme_dumps (hpdftbl_theme_t ∗theme, char ∗buff, size_t buffsize)

    Serialize theme structure to a string buffer.
- int hpdftbl_theme_loads (hpdftbl_theme_t ∗tbl, char ∗buff)

    Load theme from a serialized string. This is the invert function of hpdftbl_theme_dumps().
- int hpdftbl_theme_load (hpdftbl_theme_t ∗tbl, char ∗filename)

    Read a theme from a previous serialized theme from a named file.
- size_t xstrlcat (char ∗dst, const char ∗src, size_t siz)

    Safe string concatenation.
- size_t xstrlcpy (char ∗__restrict dst, const char ∗__restrict src, size_t dsize)

    Safe string copy.
- int hpdftbl_read_file (char ∗buff, size_t buffsize, char ∗filename)

    Read content of file into a specified buffer.
- _Bool chktbl (hpdftbl_t, size_t, size_t)

    Internal function. Check that a row and column are within the table.

Variables

- int hpdftbl_err_code

    Stores the last generated error code.
- int hpdftbl_err_row

    The row where the last error was generated.

- int hpdftbl_err_col

    The column where the last error was generated.

- int hpdftbl_err_lineno

    Hold the line number of the last error occurred.
- char ∗ hpdftbl_err_file

    Hold the file name where the last error occurred.
- char hpdftbl_err_extrainfo [ ]

    Extra info that may be specified at the point of error.
- hpdftbl_error_handler_t **hpdftbl_err_handler**

    This stores a pointer to the function acting as the error handler callback.

16.8.1 Detailed Description

Header file for libhpdftbl.

Author

> Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

> LICENSE

Released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN-CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.8.2 Macro Definition Documentation

16.8.2.1 _HPDFTBL_SET_ERR

```
#define _HPDFTBL_SET_ERR(
            t,
            err,
            r,
            c ) do {hpdftbl_err_code=err;hpdftbl_err_row=r;hpdftbl_err_col=c;hpdftbl_err_lineno=←╴
__LINE__;hpdftbl_err_file=__FILE__; if(hpdftbl_err_handler){hpdftbl_err_handler(t,r,c,err);}}
while(0)
```

Call the error handler with specified error code and table row, col where error occured.

Parameters

t	Table handler
err	Error code
r	Row where error occured
c	Column where error occured

16.8.2.2 _HPDFTBL_SET_ERR_EXTRA

```
#define _HPDFTBL_SET_ERR_EXTRA(
            info ) do {strncpy(hpdftbl_err_extrainfo,info,1023);hpdftbl_err_extrainfo[1023]=0;}
while(0)
```

Set optional extra info at error state. (Currently only used by the late binding setting callback functions)

Parameters

info	Extra info that can be set by a function at a state of error

See also

> hpdftbl_set_label_dyncb(),hpdftbl_set_content_dyncb()

16.8.2.3 DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR

```
#define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5
```

The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size.

The margin is calculated as:
```
bottom_margin = fontsize * AUTO_VBOTTOM_MARGIN_FACTOR
```

See also

> hpdftbl_set_bottom_vmargin_bottom()

16.8.2.4 HPDF_FF_COURIER

```
#define HPDF_FF_COURIER "Courier"
```

Font family

16.8.2.5 HPDF_FF_COURIER_BOLD

```
#define HPDF_FF_COURIER_BOLD "Courier-Bold"
```

Font family

Examples

> example01.c.

16.8.2.6 HPDF_FF_COURIER_BOLDITALIC

`#define HPDF_FF_COURIER_BOLDITALIC "Courier-BoldOblique"`

Font family

16.8.2.7 HPDF_FF_COURIER_IALIC

`#define HPDF_FF_COURIER_IALIC "Courier-Oblique"`

Font family

16.8.2.8 HPDF_FF_HELVETICA

`#define HPDF_FF_HELVETICA "Helvetica"`

Font family

16.8.2.9 HPDF_FF_HELVETICA_BOLD

`#define HPDF_FF_HELVETICA_BOLD "Helvetica-Bold"`

Font family

Examples

example01.c, and tut_ex09.c.

16.8.2.10 HPDF_FF_HELVETICA_BOLDITALIC

`#define HPDF_FF_HELVETICA_BOLDITALIC "Helvetica-BoldOblique"`

Font family

16.8.2.11 HPDF_FF_HELVETICA_ITALIC

`#define HPDF_FF_HELVETICA_ITALIC "Helvetica-Oblique"`

Font family

16.8.2.12 HPDF_FF_TIMES

```
#define HPDF_FF_TIMES "Times-Roman"
```

Font family

Examples

[tut_ex09.c.](#)

16.8.2.13 HPDF_FF_TIMES_BOLD

```
#define HPDF_FF_TIMES_BOLD "Times-Bold"
```

Font family

16.8.2.14 HPDF_FF_TIMES_BOLDITALIC

```
#define HPDF_FF_TIMES_BOLDITALIC "Times-BoldItalic"
```

Font family

16.8.2.15 HPDF_FF_TIMES_ITALIC

```
#define HPDF_FF_TIMES_ITALIC "Times-Italic"
```

Font family

16.8.2.16 hpdftbl_cm2dpi

```
#define hpdftbl_cm2dpi(
              c ) (((HPDF_REAL)(c))/2.54*72)
```

Convert cm to dots using the default resolution (72 DPI)

Parameters

c	Measure in cm

Returns

HPDF_REAL Converted value in dots

Examples

[example01.c,](#) [tut_ex00.c,](#) [tut_ex01.c,](#) [tut_ex02.c,](#) [tut_ex02_1.c,](#) [tut_ex03.c,](#) [tut_ex04.c,](#) [tut_ex05.c,](#) [tut_ex06.c,](#)

16.8.2.17 TABLE_JSON_VERSION

`#define TABLE_JSON_VERSION 1`

Data structure version for serialization of tables

16.8.2.18 THEME_JSON_VERSION

`#define THEME_JSON_VERSION 1`

Data structure version for serialization of themes

16.8.3 Typedef Documentation

16.8.3.1 hpdf_text_style_t

`typedef struct text_style hpdf_text_style_t`

Specification of a text style.

This structure collects the basic properties for a text string (font, color, background, horizontal alignment)

16.8.3.2 hpdftbl_callback_t

`typedef void(* hpdftbl_callback_t) (hpdftbl_t)`

Callback type for optional post processing when constructing table from a data array.

Type for generic table callback used when constructing a table from data. This can be used to perform any potential table manipulation. The callback happens after the table has been fully constructed and just before it is stroked.

See also

hpdftbl_stroke_from_data()

16.8.3.3 hpdftbl_canvas_callback_t

```
typedef void(* hpdftbl_canvas_callback_t) (HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_↩
REAL, HPDF_REAL, HPDF_REAL, HPDF_REAL)
```

Type specification for the table canvas callback.

A canvas callback, if specified, is called for each cell before the content is stroked. The callback will be given the bounding box for the cell (x,y,width,height) in addition to the row and column the cell has.

See also

hpdftbl_set_canvas_cb()

16.8.3.4 hpdftbl_cell_spec_t

```
typedef struct hpdftbl_cell_spec hpdftbl_cell_spec_t
```

Used in data driven table creation.

A table can be specified by creating a array of this structure together with the hpdftbl_spec_t structure. The array should have one entry for each cell in the table.

See also

hpdftbl_stroke_from_data()

16.8.3.5 hpdftbl_cell_t

```
typedef struct hpdftbl_cell hpdftbl_cell_t
```

Type definition for the cell structure.

This is an internal structure that represents an individual cell in the table.

16.8.3.6 hpdftbl_content_callback_t

```
typedef char *(* hpdftbl_content_callback_t) (void *, size_t, size_t)
```

Type specification for the table content callback.

The content callback is used to specify the textual content in a cell and is an alternative method to specifying the content to be displayed.

See also

hpdftbl_set_content_cb()

16.8.3.7 hpdftbl_content_style_callback_t

```
typedef _Bool(* hpdftbl_content_style_callback_t) (void *, size_t, size_t, char *content, hpdf_text_style_t
*)
```

Type specification for the content style.

The content callback is used to specify the textual style in a cell and is an alternative method to specifying the style of content to be displayed.

See also

> hpdftbl_set_content_style_cb()

16.8.3.8 hpdftbl_error_handler_t

```
typedef void(* hpdftbl_error_handler_t) (hpdftbl_t, int, int, int)
```

TYpe for error handler function.

The error handler (of set) will be called if the table library descovers an error condition

See also

> hpdftbl_set_errhandler()

16.8.3.9 hpdftbl_grid_style_t

```
typedef struct grid_style hpdftbl_grid_style_t
```

Specification for table grid lines.

Contains line properties used when stroking a grid line

16.8.3.10 hpdftbl_line_dashstyle_t

```
typedef enum hpdftbl_dashstyle hpdftbl_line_dashstyle_t
```

Possible line dash styles for grid lines.

In the illustration of the patterns "x"=solid and "_"=space.

For each pattern we show two full cycles which should give a good visual indication of the different patterns.

16.8.3.11 hpdftbl_spec_t

`typedef struct hpdftbl_spec hpdftbl_spec_t`

Used in data driven table creation.

This is used together with an array of cell specification hpdftbl_cell_spec_t to specify the layout of a table.

16.8.3.12 hpdftbl_t

`typedef struct hpdftbl* hpdftbl_t`

Table handle is a pointer to the hpdftbl structure.

This is the basic table handle used in almost all API calls. A table reference is returned when a table is created.

See also

hpdftbl_create()

16.8.3.13 hpdftbl_text_align_t

`typedef enum hpdftbl_text_align hpdftbl_text_align_t`

Enumeration for horizontal text alignment.

See also

hpdftbl_set_header_halign()

hpdftbl_set_title_halign()

hpdftbl_text_align

16.8.3.14 hpdftbl_theme_t

`typedef struct hpdftbl_theme hpdftbl_theme_t`

Define a set of styles into a table theme.

Contains all information about the styles of various elements in the table that together make up the table style

16.8.4 Enumeration Type Documentation

16.8.4.1 hpdftbl_dashstyle

`enum hpdftbl_dashstyle`

Possible line dash styles for grid lines.

In the illustration of the patterns "x"=solid and "_"=space.

For each pattern we show two full cycles which should give a good visual indication of the different patterns.

Enumerator

LINE_SOLID	Solid line
LINE_DOT1	Dotted line variant 1 "x_x_x_"
LINE_DOT2	Dotted line variant 2 "x__x__x__"
LINE_DOT3	Dotted line variant 3 "x___x___x___"
LINE_DOT4	Dotted line variant 3 "x_____x_____x_____"
LINE_DASH1	Dashed line variant 1 "xx__xx__xx__"
LINE_DASH2	Dashed line variant 2 "xx___xx___xx___"
LINE_DASH3	Dashed line variant 3 "xxxx__xxxx__xxxx__"
LINE_DASH4	Dashed line variant 4 "xxxx____xxxx____xxxx____"
LINE_DASH5	Dashed line variant 4 "xxxxxx___xxxxxx___xxxxxx___"
LINE_DASHDOT1	Dashed-dot line variant 1 "xxxxx__xx__xxxxx__xx__xxxxx__xx__"
LINE_DASHDOT2	Dashed-dot line variant 1 "xxxxxxx___xxx___xxxxxxx___xxx___xxxxxxx___xxx___"

16.8.4.2 hpdftbl_text_align

enum [hpdftbl_text_align](#)

Enumeration for horizontal text alignment.

See also

> [hpdftbl_set_header_halign()](#)
>
> [hpdftbl_set_title_halign()](#)
>
> [hpdftbl_text_align](#)

Enumerator

LEFT	Left test alignment
CENTER	Center test alignment
RIGHT	Right test alignment

16.8.5 Function Documentation

16.8.5.1 chktbl()

```
_Bool chktbl (
            hpdftbl_t t,
            size_t r,
            size_t c )
```

Internal function. Check that a row and column are within the table.

Internal function. Check that a row and column are within the table

Parameters

t	Table handle
r	Row
c	Column

Returns

TRUE if within bounds, FALSE otherwise

Referenced by hpdftbl_set_cell(), hpdftbl_set_cell_canvas_cb(), hpdftbl_set_cell_content_cb(), hpdftbl_set_cell_content_style(), hpdftbl_set_cell_content_style_cb(), hpdftbl_set_cell_label_cb(), and hpdftbl_set_cellspan().

16.8.5.2 HPDF_RoundedCornerRectangle()

```
void HPDF_RoundedCornerRectangle (
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            HPDF_REAL width,
            HPDF_REAL height,
            HPDF_REAL rad )
```

Draw rectangle with rounded corner.

Draw a rectangle with rounded corner with the current line width, color. The rectangle will not be stroked.

Parameters

page	Page handle
xpos	Lower left x-position of rectangle
ypos	Lower left y-position of rectangle
width	Width of rectangle
height	Height of rectangle
rad	Radius of corners

Referenced by hpdftbl_widget_slide_button().

16.8.5.3 hpdftbl_apply_theme()

```
int hpdftbl_apply_theme (
            hpdftbl_t t,
            hpdftbl_theme_t * theme )
```

Apply a specified theme to a table.

Note however that a limitation (by design) of themes is that settings in individual cells are not recorded in a theme since a theme can be applied to any table despite the structure. This mean only settings that are generic to a table is stored in a theme. Not individal cells.

The default table theme can be retrieved with hpdftbl_get_default_theme()

Parameters

t	Table handle
theme	Theme reference

Returns

0 on success, -1 on failure

See also

hpdftbl_get_default_theme()

Examples

tut_ex41.c.

Referenced by hpdftbl_create_title(), and hpdftbl_stroke_from_data().

16.8.5.4 hpdftbl_clear_spanning()

```
int hpdftbl_clear_spanning (
            hpdftbl_t t )
```

Clear all cell spanning.

Reset all spanning cells to no spanning

Parameters

t	Table handle

Returns

0 on success, -1 on failure

See also

hpdftbl_set_cellspan()

16.8.5.5 hpdftbl_create()

```
hpdftbl_t hpdftbl_create (
            size_t rows,
            size_t cols )
```

Create a new table with no title.

Create a new table structure. This is the basic handler needed for most other API functions.

Parameters

rows	Number of rows
cols	Number of columns

Returns

>A handle to a table, NULL in case of OOM

16.8.5.6 hpdftbl_create_title()

```
hpdftbl_t hpdftbl_create_title (
            size_t rows,
            size_t cols,
            char * title )
```

Create a new table with title top row.

Create a new table structure. This is the basic handler needed for most other API functions.

Parameters

rows	Number of rows
cols	Number of columns
title	Title of table

Returns

>A handle to a table, NULL in case of OOM

Referenced by hpdftbl_create(), and hpdftbl_stroke_from_data().

16.8.5.7 hpdftbl_default_table_error_handler()

```
void hpdftbl_default_table_error_handler (
            hpdftbl_t t,
```

```
              int r,
              int c,
              int err )
```

A basic default table error handler.

This error handler is used as a callback that outputs the error to stderr in human readable format and quits the process.

Parameters

t	Table where the error happened (can be NULL)
r	Cell row
c	Cell column
err	The error code

See also

> [hpdftbl_set_errhandler()](#)

Examples

> [tut_ex10.c](#), [tut_ex11.c](#), and [tut_ex12.c](#).

16.8.5.8 hpdftbl_destroy()

```
int hpdftbl_destroy (
              hpdftbl_t t )
```

Destroy a table and free all memory.

Destroy a table previous created with table_create(), It is the calling routines responsibility not to acceess t again.

Parameters

t	Handle to table

Returns

> 0 on success, -1 on failure

Referenced by [hpdftbl_stroke_from_data()](#).

16.8.5.9 hpdftbl_destroy_theme()

```
int hpdftbl_destroy_theme (
              hpdftbl_theme_t * theme )
```

Destroy existing theme structure and free memory.

Free all memory allocated by a theme

Parameters

theme	The theme to free

Returns

-1 for error , 0 for success

Examples

example01.c.

Referenced by hpdftbl_create_title().

16.8.5.10 hpdftbl_dump()

```
int hpdftbl_dump (
            hpdftbl_t tbl,
            char ∗ filename )
```

Serialize a table structure as a JSON file.

The table is serialized as JSON file and have whitespaces and newlines to make it more human readable. The serialization is a complete representation of a table.

Parameters

tbl	Table handle
filename	Filename to write to. Any path specified must exists

Returns

-1 on failure, 0 on success

Examples

tut_ex40.c, and tut_ex41.c.

16.8.5.11 hpdftbl_dumps()

```
int hpdftbl_dumps (
            hpdftbl_t tbl,
            char ∗ buff,
            size_t buffsize )
```

Serialize a table structure to a string buffer.

The table is serialized as JSON and have whitespaces and newlines to make it more human readable. Note is is the callers responsibility to make sure the buffer is large enough to hold the serialized table.

Parameters

tbl	Table handle of table to dump
buff	Buffer to dump structure to
buffsize	Size of buffer

Returns

-1 on failure, 0 on success

See also

hpdftbl_load(),hpdftbl_dump(), hpdftbl_stroke_pos(),

Referenced by hpdftbl_dump().

16.8.5.12 hpdftbl_encoding_text_out()

```
int hpdftbl_encoding_text_out (
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            char * text )
```

Strke text with current encoding.

Utility function to stroke text with character encoding. It is the calling routines responsibility to enclose text in a HPDF_Page_BeginText() / HPDF_Page_EndText()

Parameters

page	Page handle
xpos	X coordinate
ypos	Y coordinate
text	Text to print

Returns

-1 on error, 0 on success

16.8.5.13 hpdftbl_get_anchor_top_left()

```
_Bool hpdftbl_get_anchor_top_left (
            hpdftbl_t tbl )
```

Get stroking anchor point.

Get anchor point for table positioning. By default the top left is used.

Parameters

tbl	Table handle

See also

hpdftbl_set_anchor_top_left

Returns

TRUE if anchor is top left, FALSE otherwise

16.8.5.14 hpdftbl_get_default_theme()

```
hpdftbl_theme_t * hpdftbl_get_default_theme (
            void  )
```

Return the default theme.

Create and return a theme corresponding to the default table theme. It is the calling functions responsibility to call hpdftbl_destroy_theme() to free the allocated memory. The default theme is a good starting point to just make minor modifications without having to define all elements.

Returns

A new theme initialized to the default settings. It is the calling routines responsibility to free memory used in the returned theme with hpdftbl_destroy_theme()

See also

hpdftbl_apply_theme(), hpdftbl_destroy_theme()

Examples

example01.c, and tut_ex41.c.

Referenced by hpdftbl_create_title().

16.8.5.15 hpdftbl_get_errstr()

```
const char * hpdftbl_get_errstr (
            int err )
```

Translate a table error code to a human readable string.

The function returns a pointer to a static string that cannot be modified. It will translate both internal table error messages as well as generic HPDF library error codes.

Parameters

err	The error code to be translated

Returns

Static pointer to string for valid error code, NULL otherwise

See also

hpdftbl_hpdf_get_errstr()

Referenced by hpdftbl_default_table_error_handler(), and hpdftbl_get_last_errcode().

16.8.5.16 hpdftbl_get_last_auto_height()

```
int hpdftbl_get_last_auto_height (
            HPDF_REAL * height )
```

Get the height calculated for the last constructed table.

Get the last automatically calculated height when stroking a table. (The height will be automatically calculated if it was specified as 0)

Parameters

height	Returned height

Returns

-1 on error, 0 if successful

16.8.5.17 hpdftbl_get_last_err_file()

```
void hpdftbl_get_last_err_file (
            int * lineno,
            char ** file,
            char ** extrainfo )
```

Get the filename and line number where the last error occurred.

Parameters

lineno	Set to the line number where the error occurred
file	Set to the file where the error occurred
extrainfo	Extra info string that may be set at the point of error

16.8.5.18 hpdftbl_get_last_errcode()

```
int hpdftbl_get_last_errcode (
            const char ** errstr,
            int * row,
            int * col )
```

Return last error code.

Return last error code. if errstr is not NULL a human readable string describing the error will be copied to the string. The error code will be reset after call.

Parameters

errstr	A string buffer where the error string is written to
row	The row where the error was found
col	The col where the error was found

Returns

> The last error code

Examples

> example01.c.

16.8.5.19 hpdftbl_get_theme()

```
int hpdftbl_get_theme (
            hpdftbl_t tbl,
            hpdftbl_theme_t * theme )
```

Extract theme from settings of a specific table.

This is useful if a table has been specified with some specific look & feel and another table should be given the same l&f.

Note however that a limitation (by design) of themes is that settings in individual cells are not recorded in a theme since a theme can be applied to any table despite the structure. This mean only settings that are generic to a table is stored in a theme. Not individal cells.

Parameters

tbl	Table handle for table to have its settings extracted
theme	Theme to be read out to.

Returns

0 on success, -1 on failure

Examples

tut_ex41.c.

16.8.5.20 hpdftbl_hpdf_get_errstr()

```
const char * hpdftbl_hpdf_get_errstr (
            const HPDF_STATUS err_code )
```

Function to return a human readable error string for an error code from Core HPDF library.

The various error codes given by the HPDF library can be translated back to a string by the usage of this function. The function will return a pointer to a static string that can not be manipulated.

Parameters

err_code	The error code

Returns

A pointer to an error string, NULL if the error code is invalid

See also

hpdftbl_get_errstr()

16.8.5.21 hpdftbl_load()

```
int hpdftbl_load (
            hpdftbl_t tbl,
            char * filename )
```

Import a table structure from a serialized table on file.

The json file make it possible to adjust the table directly. However it is easy to get it wrong. Some things to keep in mind while doing manual changes.

- A real number must always be written as a decimal number with at least one decimal point (even if it .0)

- Remember that the width of the table is specified manually and not automatically recalculated based on the text width.

After reading a serialized table it can asily be be stroked with only two lines of code as the following code-snippet shows

```
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
if(0 == hpdftbl_load(tbl, "mytablefile.json") ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

Note

>   The `hpdftbl_t` is a pointer type to `struct hpdftbl` and hence must must be either dynamically allocated as the example here shows or an instance of the struct must be created whose address is given to this functions.

Parameters

tbl	Table to read into
filename	File to read from

Returns

>   0 on success, -1 on file parse error, -2 on nay other error

Examples

>   tut_ex40.c, and tut_ex41.c.

16.8.5.22 hpdftbl_loads()

```
int hpdftbl_loads (
            hpdftbl_t tbl,
            char * buff )
```

Import a table structure from a serialized json buffert.

This is the preferred way on how to store a table structure in for example a database.

Example:
```
char *mybuffer = ....
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
if(0 == hpdftbl_load(tbl, mybuffer) ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

Parameters

tbl	Reference to table handle to be populated
buff	Buffer with serialized data to read back

Returns

>   0 on success, -1 on file parse error, -2 on nay other error

See also

>   hpdftbl_dump(), hpdftbl_load(), hpdftbl_stroke_pos()

Referenced by hpdftbl_load().

16.8.5.23 hpdftbl_read_file()

```
int hpdftbl_read_file (
            char * buff,
            size_t buffsize,
            char * filename )
```

Read content of file into a specified buffer.

Parameters

buff	Destination buffer
buffsize	Size of buffer
filename	Name of file to read from

Returns

-1 on failure, 0 on success

Referenced by hpdftbl_load(), and hpdftbl_theme_load().

16.8.5.24 hpdftbl_set_anchor_top_left()

```
void hpdftbl_set_anchor_top_left (
            hpdftbl_t tbl,
            const _Bool anchor )
```

Switch stroking anchor point.

Set anchor point for table positioning. By default the top left is used as anchor. Calling this function with FALSE can sets the anchor to bottom left instead.

Parameters

tbl	Table handle
anchor	Set to TRUE to use top left as anchor, FALSE for bottom left

16.8.5.25 hpdftbl_set_background()

```
int hpdftbl_set_background (
            hpdftbl_t t,
            HPDF_RGBColor background )
```

Set table background color.

Parameters

t	Table handle
background	Background color

Returns

0 on success, -1 on failure

16.8.5.26   hpdftbl_set_bottom_vmargin_factor()

```
void hpdftbl_set_bottom_vmargin_factor (
            hpdftbl_t t,
            HPDF_REAL f )
```

The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:
```
bottom_margin = fontsize * f
```

The default margin is specified by the define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR

Parameters

t	Table handle
f	Bottom margin factor

Referenced by hpdftbl_apply_theme().

16.8.5.27   hpdftbl_set_canvas_cb()

```
int hpdftbl_set_canvas_cb (
            hpdftbl_t t,
            hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set cell canvas callback. This callback gets called for each cell in the table. The purpose is to allow the client to add dynamic content to the specified cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell. To set the canvas callback only for a specific cell use the hpdftbl_set_cell_canvas_cb() function

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_set_cell_canvas_cb()

16.8.5.28 hpdftbl_set_canvas_dyncb()

```
int hpdftbl_set_canvas_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as canvas callback. This function must follow the signature of a callback function as specified in hpdftbl_canvas_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_canvas_cb(), hpdftbl_canvas_callback_t

16.8.5.29 hpdftbl_set_cell()

```
int hpdftbl_set_cell (
            hpdftbl_t t,
            size_t r,
            size_t c,
            char * label,
            char * content )
```

Set content for specific cell.

Set label and content for a specific cell. If the specified cell is part of another cells spanning an error occurs (returns -1),

Parameters

t	Table handle
r	Row
c	Column
label	Label
content	Text content

Returns

-1 on error, 0 if successful

Referenced by hpdftbl_stroke_from_data().

16.8.5.30 hpdftbl_set_cell_canvas_cb()

```
int hpdftbl_set_cell_canvas_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set a canvas callback for an individual cell. This will override the table canvas callback. The canvas callback is called with arguments that give the bounding box for the cell. In that way a callback function may draw arbitrary graphic in the cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_canvas_callback_t

hpdftbl_set_canvas_cb()

Examples

example01.c, and tut_ex14.c.

Referenced by hpdftbl_stroke_from_data().

16.8.5.31   hpdftbl_set_cell_canvas_dyncb()

```
int hpdftbl_set_cell_canvas_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback cell canvas function.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
r	Celll row
c	Cell column
cb_name	Name of the function to be used as canvas callback. This function must follow the signature of a callback function as specified in hpdftbl_canvas_callback_t

Returns

See also

> hpdftbl_set_cell_canvas_cb(), hpdftbl_canvas_callback_t

16.8.5.32   hpdftbl_set_cell_content_cb()

```
int hpdftbl_set_cell_content_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_content_callback_t cb )
```

Set cell content callback.

Set a content callback for an individual cell. This will override the table content callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

t	Table handle
cb	Callback function
r	Cell row
c	Cell column

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_set_content_cb()

Examples

tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex14.c.

Referenced by hpdftbl_stroke_from_data().

16.8.5.33 hpdftbl_set_cell_content_dyncb()

```
int hpdftbl_set_cell_content_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback cell content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb_name	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_content_cb(), hpdftbl_content_callback_t

Examples

tut_ex30.c.

16.8.5.34 hpdftbl_set_cell_content_style()

```
int hpdftbl_set_cell_content_style (
            hpdftbl_t t,
            size_t r,
            size_t c,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for content of specified cell.

SSet the font style for content of specified cell. This will override the global cell content setting.

Parameters

t	Table handle
r	Cell row
c	Cell column
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

0 on success, -1 on failure

See also

hpdftbl_set_content_style()

hpdftbl_set_cell_content_style_cb()

Referenced by hpdftbl_set_col_content_style(), and hpdftbl_set_row_content_style().

16.8.5.35 hpdftbl_set_cell_content_style_cb()

```
int hpdftbl_set_cell_content_style_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_content_style_callback_t cb )
```

Set cell specific callback to specify cell content style.

Set callback to format the style for the specified cell

Parameters

t	Table handle
r	Cell row
c	Cell column
cb	Callback function

Returns

0 on success, -1 on failure

See also

hpdftbl_set_ontent_style_cb()

Referenced by hpdftbl_stroke_from_data().

16.8.5.36 hpdftbl_set_cell_content_style_dyncb()

```
int hpdftbl_set_cell_content_style_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback for cell style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by hpdftbl_content_style_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_style_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_cell_content_style_cb(), hpdftbl_content_style_callback_t

16.8.5.37 hpdftbl_set_cell_label_cb()

```
int hpdftbl_set_cell_label_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_content_callback_t cb )
```

Set cell label callback.

Set a label callback for an individual cell. This will override the table label callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

t	Table handle
cb	Callback function
r	Cell row
c	Cell column

Returns

    -1 on failure, 0 otherwise

See also

    hpdftbl_set_label_cb()

Referenced by hpdftbl_stroke_from_data().

16.8.5.38 hpdftbl_set_cell_label_dyncb()

```
int hpdftbl_set_cell_label_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback for cell label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by hpdftbl_content_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_cell_label_cb(), hpdftbl_content_callback_t

16.8.5.39 hpdftbl_set_cellspan()

```
int hpdftbl_set_cellspan (
            hpdftbl_t t,
            size_t r,
            size_t c,
            size_t rowspan,
            size_t colspan )
```

Set cell spanning.

Set row and column spanning for a cell, an expanded cell is referenced via the position of it's top-left cell

Parameters

t	Table handle
r	Row
c	Column
rowspan	Row span
colspan	Column span

Returns

-1 on error, 0 if successful

See also

hpdftbl_clear_spanning()

Referenced by hpdftbl_stroke_from_data().

16.8.5.40 hpdftbl_set_col_content_style()

```
int hpdftbl_set_col_content_style (
            hpdftbl_t t,
            size_t c,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for an entire column of cells.

Set font options for the specified column of cells. This will override the global cell content setting.

Parameters

t	Table handle
c	Column to affect
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

0 on success, -1 on failure

See also

hpdftbl_set_content_style()

hpdftbl_set_cell_content_style_cb()

16.8.5.41 hpdftbl_set_colwidth_percent()

```
int hpdftbl_set_colwidth_percent (
            hpdftbl_t t,
            size_t c,
            float w )
```

Set column width as percentage of overall table width.

Specify column width as percentage of total column width. Note that this will only take effect if the table has an overall width specified when stroked. Too avoid errors one column should be left unspecified to let the library use whatever space is left for that column.

Parameters

t	Table handle
c	Column to set width of first column has index 0
w	Width as percentage in range [0.0, 100.0]

Returns

0 on success, -1 on failure

16.8.5.42 hpdftbl_set_content()

```
int hpdftbl_set_content (
            hpdftbl_t t,
            char ** content )
```

Set the content for the table.

Set content for all cells. It is the calling functions responsibility that the content array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r ∗ num_cols + c) where num_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with strdup() so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N∗M) entries.

Another way to specify the content is to use the callback mechanism. By setting up a content callback function that returns the content for a cell.

Parameters

t	Table handle
content	A one dimensional string array of content string

Returns

-1 on error, 0 if successful

See also

hpdftbl_set_content_callback()

hpdftbl_set_cell_content_callback()

16.8.5.43 hpdftbl_set_content_cb()

```
int hpdftbl_set_content_cb (
            hpdftbl_t t,
            hpdftbl_content_callback_t cb )
```

Set table content callback.

This callback gets called for each cell in the table and the returned string will be used as the content. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the content will be set to the content specified with the direct content setting. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

t	Table handle
cb	Callback function

Returns

-1 for error , 0 otherwise

See also

　　hpdftbl_set_cell_content_cb()

Examples

　　tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex09.c.

Referenced by hpdftbl_stroke_from_data().

16.8.5.44   hpdftbl_set_content_dyncb()

```
int hpdftbl_set_content_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

　　-1 on failure, 0 on success

See also

　　hpdftbl_set_content_cb(), hpdftbl_content_callback_t

Examples

　　tut_ex30.c.

16.8.5.45 hpdftbl_set_content_style()

```
int hpdftbl_set_content_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set text style for text content.

Set style options for cell content (font, color, background). This will be applied for all cells in the table. If a style callback have been specified for either the table or a cell that style take precedence.

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

-1 on error, 0 if successful

See also

hpdftbl_set_cell_content_style()

hpdftbl_set_cell_content_style_cb()

Referenced by hpdftbl_apply_theme().

16.8.5.46 hpdftbl_set_content_style_cb()

```
int hpdftbl_set_content_style_cb (
            hpdftbl_t t,
            hpdftbl_content_style_callback_t cb )
```

Set callback to specify cell content style.

Set callback to format the style for cells in the table. If a cell has its own content style callback that callback will override the generic table callback.

Parameters

t	Table handle
cb	Callback function

Returns

0 on success, -1 on failure

See also

[hpdftbl_set_cell_content_style_cb()](#)

Examples

[tut_ex09.c.](#)

Referenced by [hpdftbl_stroke_from_data().](#)

16.8.5.47 hpdftbl_set_content_style_dyncb()

```
int hpdftbl_set_content_style_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback for table style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by hpdftbl_content_style_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file()](#) to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_style_callback_t.

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_content_style_cb(),](#) [hpdftbl_content_style_callback_t](#)

16.8.5.48 hpdftbl_set_dlhandle()

```
void hpdftbl_set_dlhandle (
            void * handle )
```

Set the handle for scope of dynamic function search.

When using late binding (some os _dyncb() functions) the scope for where the runtime searches for the functions can be specified as is discussed in `man 3 dlsym`. By default the library uses dl_handle which make the library first searches the current image and then all images it was built against.

If the dynamic callbacks are located in a runtime loaded library then the handle returned by dlopen() must be specified as the function will not be found otherwise.

Parameters

handle	Predefined values or the handle returned by dlopen() (see man dlopen)

16.8.5.49 hpdftbl_set_errhandler()

```
hpdftbl_error_handler_t hpdftbl_set_errhandler (
            hpdftbl_error_handler_t err_handler )
```

Specify errhandler for the table routines.

Note: The library provides a basic default error handler that can be used,

Parameters

err_handler	

Returns

The old error handler or NULL if non exists

See also

hpdftbl_default_table_error_handler()

Examples

tut_ex10.c, tut_ex11.c, and tut_ex12.c.

16.8.5.50 hpdftbl_set_header_halign()

```
int hpdftbl_set_header_halign (
            hpdftbl_t t,
            hpdftbl_text_align_t align )
```

Set table header horizontal text align.

Parameters

t	Table handle
align	Alignment

Returns

0 on success, -1 on failure

Referenced by hpdftbl_apply_theme().

16.8.5.51   hpdftbl_set_header_style()

```
int hpdftbl_set_header_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Specify style for table header row.

Set the font properties and background for the header row which is the top row if enabled. The header row will be automatically enabled after calling this function. The header can be enabled/disabled separately with hpdftbl_use_header()

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Font color
background	Cell background color

Returns

0 on success, -1 on failure

See also

hpdftbl_use_header()

Referenced by hpdftbl_apply_theme().

16.8.5.52 hpdftbl_set_inner_grid_style()

```
int hpdftbl_set_inner_grid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set inner border grid style.

This is a shortform to set both the vertical and horizontal gridline style with one call.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_inner_hgrid_style(), hpdftbl_set_inner_vgrid_style(), hpdftbl_set_outer_grid_style()

16.8.5.53 hpdftbl_set_inner_hgrid_style()

```
int hpdftbl_set_inner_hgrid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set inner horizontal border grid style.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

> 0 on success, -1 on failure

See also

> [hpdftbl_set_inner_grid_style()](), [hpdftbl_set_inner_vgrid_style()]()

Referenced by [hpdftbl_apply_theme()](), and [hpdftbl_set_inner_grid_style()]().

16.8.5.54 hpdftbl_set_inner_tgrid_style()

```
int hpdftbl_set_inner_tgrid_style (
          hpdftbl_t t,
          HPDF_REAL width,
          HPDF_RGBColor color,
          hpdftbl_line_dashstyle_t dashstyle )
```

Set inner horizontal top border grid style.

This would be the gridline just below the header row.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

> 0 on success, -1 on failure

See also

> [hpdftbl_set_inner_hgrid_style()]()

Referenced by [hpdftbl_apply_theme()]().

16.8.5.55 hpdftbl_set_inner_vgrid_style()

```
int hpdftbl_set_inner_vgrid_style (
          hpdftbl_t t,
          HPDF_REAL width,
          HPDF_RGBColor color,
          hpdftbl_line_dashstyle_t dashstyle )
```

Set inner vertical border grid style.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

 0 on success, -1 on failure

See also

 hpdftbl_set_inner_grid_style(), hpdftbl_set_inner_hgrid_style()

Referenced by hpdftbl_apply_theme(), and hpdftbl_set_inner_grid_style().

16.8.5.56 hpdftbl_set_label_cb()

```
int hpdftbl_set_label_cb (
            hpdftbl_t t,
            hpdftbl_content_callback_t cb )
```

Set table label callback.

Set label callback. This callback gets called for each cell in the table and the returned string will be used as the label. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the label will be set to the content specified with the direct label setting. The callback function will receive the Table tag and the row and column

Parameters

t	Table handle
cb	Callback function

Returns

 -1 on failure, 0 otherwise

See also

 hpdftbl_content_callback_t

 hpdftbl_set_cell_label_cb()

Examples

 tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex14.c.

Referenced by hpdftbl_stroke_from_data().

16.8.5.57 hpdftbl_set_label_dyncb()

```
int hpdftbl_set_label_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback for table label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. symbols. The callback function must have the signature defined by hpdftbl_content_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_label_cb(), hpdftbl_content_callback_t

Examples

tut_ex30.c.

16.8.5.58 hpdftbl_set_label_style()

```
int hpdftbl_set_label_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the text style for labels in the entire table.

Set font, color and background options for cell labels. If a style callback have been specified for either the table or a cell that style take precedence.

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

-1 on error, 0 if successful

Referenced by hpdftbl_apply_theme().

16.8.5.59 hpdftbl_set_labels()

```
int hpdftbl_set_labels (
            hpdftbl_t t,
            char ** labels )
```

Set the text for the cell labels.

Set labels for all the cell. It is the calling functions responsibility that the labels array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r $*$ num_cols + c) where num_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with strdup() so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N$*$M) entries.

Parameters

t	Table handle
labels	A one dimensional string array of labels

Returns

-1 on error, 0 if successful

See also

hpdftbl_set_cell_label_cb()

hpdftbl_set_label_cb()

16.8.5.60 hpdftbl_set_min_rowheight()

```
int hpdftbl_set_min_rowheight (
            hpdftbl_t t,
            float h )
```

Set the minimum row height in the table.

The row height is normally calculated based on the font size and if labels are displayed or not. However, it is not possible for the table to know the height of specific widgets (for example) without a two-pass table drawing algorithm.

To handle thos odd cases when the calculated height is not sufficient a manual minimum height can be specified.

Parameters

t	Table handler
h	The minimum height (in points). If specified as 0 the min height will have no effect.

Returns

> 0 on success, -1 on failure

16.8.5.61 hpdftbl_set_outer_grid_style()

```
int hpdftbl_set_outer_grid_style (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color,
            hpdftbl_line_dashstyle_t dashstyle )
```

Set outer border grid style.

Parameters

t	Table handle
width	Line width (in pt)
color	Line color
dashstyle	Line dash style

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_inner_grid_style()

Referenced by hpdftbl_apply_theme().

16.8.5.62 hpdftbl_set_post_cb()

```
int hpdftbl_set_post_cb (
            hpdftbl_t t,
            hpdftbl_callback_t cb )
```

Set table post processing callback.

This is an optional post processing callback for anything in general to do after the table has been constructed. The callback happens after the table has been fully constructed and just before it is stroked.

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

[hpdftbl_callback_t](#)

16.8.5.63 hpdftbl_set_post_dyncb()

```
int hpdftbl_set_post_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Set table post processing callback.

This is an optional post processing callback for anything in general to do after the table has been constructed. The callback only gets the table as its first and only argument. The callback happens after the table has been fully constructed and just before it is stroked.

Parameters

t	Table handle
cb_name	Callback function name

Returns

-1 on failure, 0 otherwise

See also

[hpdftbl_callback_t](#), [hpdftbl_set_post_cb()](#)

16.8.5.64 hpdftbl_set_row_content_style()

```
int hpdftbl_set_row_content_style (
            hpdftbl_t t,
            size_t r,
            char * font,
```

```
             HPDF_REAL fsize,
             HPDF_RGBColor color,
             HPDF_RGBColor background )
```

Set the text style for an entire row of cells.

Set font options for the specified row of cells. This will override the global cell content.

Parameters

t	Table handle
r	Row to affect
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

> 0 on success, -1 on failure

See also

> [hpdftbl_set_content_style()](#)
>
> [hpdftbl_set_cell_content_style_cb()](#)

16.8.5.65 hpdftbl_set_tag()

```
int hpdftbl_set_tag (
            hpdftbl_t t,
            void * tag )
```

Set an optional tag for the table.

Set an optional tag in the table. The tag can be a pointer to anything. The tag is passed as the first argument in the various callbacks and can be used to supply table specific information or identify a specific table in the case the same callback is used for multiple tables.

Parameters

t	The table handle
tag	The tag (pointer to any object)

Returns

> 0 on success, -1 on failure

16.8.5.66 hpdftbl_set_text_encoding()

```
void hpdftbl_set_text_encoding (
            char * target,
            char * source )
```

Determine text source encoding.

The default HPDF encoding is a standard PDF encoding. The problem with that is that now almost 100% of all code is written in UTF-8 encoding and trying to print text strings with accented characters will simply not work. For example the default encoding assumes that strings are given in UTF-8 and sets the target to ISO8859-4 which includes northern europe accented characters. The conversion is internally handled by the standard iconv() routines.

Parameters

target	The target encoding. See HPDF documentation for supported encodings.
source	The source encodings, i.e. what encodings are sth strings in the source specified in.

16.8.5.67 hpdftbl_set_title()

```
int hpdftbl_set_title (
            hpdftbl_t t,
            char * title )
```

Set table title.

Set table title. A title will occupy a separate row above the table that is not included in the row count. A table is enabled when the table text is $<>$ NULL and disabled when the title text is == NULL.

Parameters

t	Table handle
title	Title string

Returns

0 on success, -1 on failure

See also

hpdftbl_set_title_style()

hpdftbl_set_title_halign()

16.8.5.68 hpdftbl_set_title_halign()

```
int hpdftbl_set_title_halign (
            hpdftbl_t t,
            hpdftbl_text_align_t align )
```

Set horizontal alignment for table title.

Parameters

t	Table handle
align	Alignment

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_title()
>
> hpdftbl_set_title_style()

Referenced by hpdftbl_apply_theme().

16.8.5.69 hpdftbl_set_title_style()

```
int hpdftbl_set_title_style (
            hpdftbl_t t,
            char * font,
            HPDF_REAL fsize,
            HPDF_RGBColor color,
            HPDF_RGBColor background )
```

Set the table title text style.

Set font options for title

Parameters

t	Table handle
font	Font name
fsize	Font size
color	Color
background	Background color

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_set_title()
>
> hpdftbl_set_title_halign()

Referenced by hpdftbl_apply_theme().

16.8.5.70 hpdftbl_set_zebra()

```
int hpdftbl_set_zebra (
            hpdftbl_t t,
            _Bool use,
            int phase )
```

Parameters

t	Table handle
use	TRUE=Use Zebra, FALSE=Don't use zebra
phase	0=Start with color 1, 1=Start with color 1

Returns

> 0 on successes -1 on failure

Referenced by hpdftbl_apply_theme().

16.8.5.71 hpdftbl_set_zebra_color()

```
int hpdftbl_set_zebra_color (
            hpdftbl_t t,
            HPDF_RGBColor z1,
            HPDF_RGBColor z2 )
```

Specify first and second color for a zebra grid table.

By default the colors start with `z1` color. To have the top row (below any potential header row) instead start with z2 specify `phase=1` in the hpdftbl_set_zebra() function.

Parameters

t	Table handle
z1	Color 1
z2	Color 2

Returns

> 0 on successes -1 on failure

Referenced by hpdftbl_apply_theme().

16.8.5.72 hpdftbl_setpos()

```
int hpdftbl_setpos (
            hpdftbl_t t,
```

```
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            HPDF_REAL height )
```

Set size and position for table.

The position is by default specified as the upper left corner of the table. Use the hpdftbl_set_origin_top_left() to use the bottom left of the table as reference point.

Ths standard stroke function hpdftbl_stroke() also take the size and position as argument for ease of use but the hpdftbl_stroke_pos() do assume that the table has it's size set.

Parameters

t	Table handle
xpos	x position for table
ypos	y position for table
width	width of table
height	height of table. If the height is specified as 0 it will be automatically calculated. The calculated height can be retrieved after the table has been stroked by a call to hpdftbl_get_last_auto_height()

Returns

-1 on error, 0 if successful

See also

hpdftbl_get_last_auto_height(), hpdftbl_set_origin_top_left()

16.8.5.73   hpdftbl_stroke()

```
int hpdftbl_stroke (
            HPDF_Doc pdf,
            const HPDF_Page page,
            hpdftbl_t t,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            HPDF_REAL height )
```

Stroke the table.

Stroke the table at the specified position and size. The position is by default specified as the upper left corner of the table. Use the hpdftbl_set_origin_top_left(FALSE) to use the bottom left of the table as reference point.

Parameters

pdf	The HPDF document handle
page	The HPDF page handle
t	Table handle
xpos	x position for table
ypos	y position for table
width	width of table
height	height of table. If the height is specified as 0 it will be automatically calculated. The calculated height can be retrieved after the table has been stroked by a call to hpdftbl_get_last_auto_height()

Returns

-1 on error, 0 if successful

See also

[hpdftbl_get_last_auto_height()](#)

[hpdftbl_stroke_from_data()](#)

Referenced by [hpdftbl_stroke_from_data()](#), and [hpdftbl_stroke_pos()](#).

16.8.5.74 hpdftbl_stroke_from_data()

```
int hpdftbl_stroke_from_data (
            HPDF_Doc pdf_doc,
            HPDF_Page pdf_page,
            hpdftbl_spec_t * tbl_spec,
            hpdftbl_theme_t * theme )
```

Construct the table from a array specification.

Create and stroke a table specified by a data structure. This makes it easier to separate the view of the data from the model which provides the data. The intended use case is that the data structure specifies the core layout of the table together with the labels and callback functions to handle the content in each cell. Using this method to create a table also makes it much more maintainable.

Parameters

pdf_doc	The PDF overall document
pdf_page	The pageto stroke to
tbl_spec	The table specification
theme	Table theme to be applied

Returns

0 on success, -1 on failure

See also

[hpdftbl_stroke()](#)

16.8.5.75 hpdftbl_stroke_grid()

```
void hpdftbl_stroke_grid (
            HPDF_Doc pdf,
            HPDF_Page page )
```

Stroke a point grid on specified page to make it easier to position text and tables.

Parameters

pdf	Document handle
page	Page handle

16.8.5.76 hpdftbl_stroke_pdfdoc()

```
int hpdftbl_stroke_pdfdoc (
            HPDF_Doc pdf_doc,
            char * file )
```

Stroke PDF document to file with check that the directory in path exists.

Note: It is a checked error if the full path is longer than 1014 characters

Parameters

pdf_doc	Haru PDF document handle
file	Full pathname of file to write to

Returns

0 on success, -1 on failure

Referenced by stroke_to_file().

16.8.5.77 hpdftbl_stroke_pos()

```
int hpdftbl_stroke_pos (
            HPDF_Doc pdf,
            const HPDF_Page page,
            hpdftbl_t t )
```

Stroke the table using the already specified size and position within the table.

Stroke the table at the specified position and size. The position is by default specified as the upper left corner of the table. Use the hpdftbl_set_origin_top_left(FALSE) to use the bottom left of the table as reference point.

This is a convenient method to use when stroking a serialized table as the table already holds the size and position. Stroking a table read back ccan be done with just two lines of code

```
hpdftbl_t tbl = calloc(1, sizeof(struct hpdftbl));
if( 0 == hpdftbl_load(tbl, filename)  ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

Parameters

pdf	The HPDF document handle
page	The HPDF page handle
t	Table handle

Returns

-1 on error, 0 if successful

See also

hpdftbl_get_last_auto_height()

hpdftbl_setpos(), hpdftbl_stroke()

16.8.5.78 hpdftbl_table_widget_letter_buttons()

```
void hpdftbl_table_widget_letter_buttons (
            HPDF_Doc doc,
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            HPDF_REAL width,
            HPDF_REAL height,
            const HPDF_RGBColor on_color,
            const HPDF_RGBColor off_color,
            const HPDF_RGBColor on_background,
            const HPDF_RGBColor off_background,
            const HPDF_REAL fsize,
            const char * letters,
            _Bool * state )
```

Display an array of letters as a table where each letter is its own "mini" cell
and sorrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different font and fac colors.

Parameters

doc	HPDF document handle
page	HPDF page handle
xpos	X-öosition of cell
ypos	Y-Position of cell
width	Width of cell
height	Height of cell
on_color	The font color in "on" state
off_color	The font color in "off" state
on_background	The face color in "on" state
off_background	The face color in "off" state
fsize	The font size
letters	What letters to have in the boxes
state	What state each boxed letter should be (0=off, 1=pn)

Examples

example01.c.

16.8.5.79 hpdftbl_theme_dump()

```
int hpdftbl_theme_dump (
            hpdftbl_theme_t * theme,
            char * filename )
```

Serialize the specified theme structure to a named file.

The theme is serialized as JSON string array and have whitespaces and newlines to make it more human readable.

Parameters

theme	Pointer to theme structure to be serialized
filename	Filename to write to

Returns

0 on success, -1 on failure

Examples

tut_ex41.c.

16.8.5.80 hpdftbl_theme_dumps()

```
int hpdftbl_theme_dumps (
            hpdftbl_theme_t * theme,
            char * buff,
            const size_t buffsize )
```

Serialize theme structure to a string buffer.

The theme is serialized as JSON string array and have whitespaces and newlines to make it more human readable.

Parameters

theme	Theme to serialize
buff	Buffer to write serialized theme to. It should be a minimum of 2k chars.
buffsize	Buffer size (including ending string NULL)

Returns

0 on success, < 0 on failure

Referenced by hpdftbl_theme_dump().

16.8.5.81 hpdftbl_theme_load()

```
int hpdftbl_theme_load (
             hpdftbl_theme_t * theme,
             char * filename )
```

Read a theme from a previous serialized theme from a named file.

Example:
```
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
hpdftbl_theme_t theme;
 if( 0 == hpdftbl_load(tbl, "tests/tut_ex41.json") ) {
     if( 0 == hpdftbl_theme_load(&theme, "mytheme.json") ) {
         hpdftbl_apply_theme(tbl, &theme);
         hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
     }
 }
```

Parameters

theme	Theme to read into
filename	File to read from

Returns

0 on success, -1 on failure

Examples

tut_ex41.c.

16.8.5.82 hpdftbl_theme_loads()

```
int hpdftbl_theme_loads (
             hpdftbl_theme_t * theme,
             char * buff )
```

Load theme from a serialized string. This is the invert function of hpdftbl_theme_dumps().

Parameters

theme	Theme to load to.
buff	Buffer which holds the previous serialized theme

Returns

0 on success, -1 on failure

See also

hpdftbl_theme_dumps(), hpdftbl_theme_load(), hpdftbl_apply_theme()

Referenced by hpdftbl_theme_load().

16.8.5.83 hpdftbl_use_header()

```
int hpdftbl_use_header (
            hpdftbl_t t,
            _Bool use )
```

Enable/disable the interpretation of the top row as a header row.

A header row will have a different style and labels will be disabled on this row. In addition the text will be centered vertically and horizontal in the cell.

Parameters

t	Table handle
use	TRUE to enable, FALSE to disable

Returns

0 on success, -1 on failure

See also

hpdftbl_set_header_style()

Referenced by hpdftbl_stroke_from_data().

16.8.5.84 hpdftbl_use_labelgrid()

```
int hpdftbl_use_labelgrid (
            hpdftbl_t t,
            _Bool use )
```

Shorter vertical line to mark labels.

Set the usage of special grid style where the vertical grid only covers the label text and a gap to the next line. Horizontal lines are drawn as usual. The label grid style gives the table a "lighter" look.

Parameters

t	Table handle
use	TRUE to use label grid, FALSE o disable it

Returns

0 on success, -1 on failure

See also

    hpdftbl_use_labels()

Referenced by hpdftbl_stroke_from_data().

16.8.5.85 hpdftbl_use_labels()

```
int hpdftbl_use_labels (
            hpdftbl_t t,
            _Bool use )
```

Enable/Disable the use of cell labels.

By default a newly created table will not use cell labels. Enabling labels will also by default enable the special label grid style. To adjust the grid style separately us the hpdftbl_use_labelgrid() method.

Parameters

t	Table handle
use	Set to TRUE for cell labels

Returns

    0 on success, -1 on failure

See also

    hpdftbl_use_labelgrid()

Referenced by hpdftbl_stroke_from_data().

16.8.5.86 hpdftbl_widget_hbar()

```
void hpdftbl_widget_hbar (
            const HPDF_Doc doc,
            const HPDF_Page page,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            const HPDF_REAL height,
            const HPDF_RGBColor color,
            const float val,
            const _Bool hide_val )
```

Draw a horizontal partially filled bar to indicate an analog (percentage) value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

Parameters

doc	HPDF Document handle
page	HPDF Page handle
xpos	Lower left x
ypos	Lower left y
width	Width of meter
height	Height of meter
color	Fill color for bar
val	Percentage fill in range [0.0, 100.0]
hide_val	TRUE to hide the value (in percent) at the right end of the entire bar

Examples

example01.c.

16.8.5.87 hpdftbl_widget_segment_hbar()

```
void hpdftbl_widget_segment_hbar (
            const HPDF_Doc doc,
            const HPDF_Page page,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            const HPDF_REAL height,
            const size_t num_segments,
            const HPDF_RGBColor on_color,
            const double val_percent,
            const _Bool hide_val )
```

Draw a horizontal segment meter that can be used to visualize a discrete value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

Parameters

doc	HPDF Document handle
page	HPDF Page handle
xpos	Lower left x
ypos	Lower left y
width	Width of meter
height	Height of meter
num_segments	Total number of segments
on_color	Color for "on" segment
val_percent	To what extent should the bars be filled (as a value 0.0 - 1.0)
hide_val	TRUE to hide the value (in percent) at the right end of the entire bar

Examples

[example01.c,](#) and [tut_ex14.c.](#)

16.8.5.88 hpdftbl_widget_slide_button()

```
void hpdftbl_widget_slide_button (
            HPDF_Doc doc,
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            HPDF_REAL width,
            HPDF_REAL height,
            _Bool state )
```

Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.

This function can not be used directly as a canvas callback since it needs the state of the button as an argument. Instead create a simple canvas callback that determines the wanted state and then just passes on all argument to this widget function.

Parameters

doc	HPDF document handle
page	HPDF page handle
xpos	X-öosition of cell
ypos	Y-Position of cell
width	Width of cell
height	Height of cell
state	State of button On/Off

Examples

[example01.c.](#)

16.8.5.89 hpdftbl_widget_strength_meter()

```
void hpdftbl_widget_strength_meter (
            const HPDF_Doc doc,
            const HPDF_Page page,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            const HPDF_REAL height,
            const size_t num_segments,
            const HPDF_RGBColor on_color,
            const size_t num_on_segments )
```

Draw a phone strength meter.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

Parameters

doc	HPDF Document handle
page	HPDF Page handle
xpos	Lower left x
ypos	Lower left y
width	Width of meter
height	Height of meter
num_segments	Total number of segments
on_color	Color for "on" segment
num_on_segments	Number of on segments

Examples

example01.c, and tut_ex14.c.

16.8.5.90 xstrlcat()

```
size_t xstrlcat (
            char * dst,
            const char * src,
            size_t siz )
```

Safe string concatenation.

Appends src to string dst of size siz (unlike strncat, siz is the full size of dst, not space left). At most siz-1 characters will be copied. Always NUL terminates (unless siz <= strlen(dst)). Returns strlen(src) + MIN(siz, strlen(initial dst)). If retval >= siz, truncation occurred.

Taken from BSD library.

Parameters

dst	Destination buffer
src	Source buffer
siz	Max size of destination buffer including terminating NULL

Returns

The number of bytes needed to be copied. If this is > `siz` then data truncation happened.

Referenced by hpdftbl_read_file().

16.8.5.91 xstrlcpy()

```
size_t xstrlcpy (
            char *__restrict dst,
            const char *__restrict src,
            size_t dsize )
```

Safe string copy.

Copy string src to buffer dst of size dsize. At most dsize-1 chars will be copied. Always NUL terminates (unless dsize == 0). Returns strlen(src); if retval >= dsize, truncation occurred.

Taken from BSD library.

Parameters

dst	Destination string
src	Source string
dsize	Maximum size of destination

Returns

strlen(src); if retval >= dsize, truncation occurred.

16.8.6 Variable Documentation

16.8.6.1 hpdftbl_err_code

```
int hpdftbl_err_code  [extern]
```

Stores the last generated error code.

Internal variable to record last error

Referenced by hpdftbl_get_errstr(), and hpdftbl_get_last_errcode().

16.8.6.2 hpdftbl_err_col

```
int hpdftbl_err_col  [extern]
```

The column where the last error was generated.

Internal variable to record last error

16.8.6.3 hpdftbl_err_extrainfo

`char hpdftbl_err_extrainfo[]  [extern]`

Extra info that may be specified at the point of error.

Internal variable to record last error

Referenced by hpdftbl_get_last_err_file().

16.8.6.4 hpdftbl_err_file

`char* hpdftbl_err_file  [extern]`

Hold the file name where the last error occurred.

Internal variable to record last error

Referenced by hpdftbl_get_last_err_file().

16.8.6.5 hpdftbl_err_lineno

`int hpdftbl_err_lineno  [extern]`

Hold the line number of the last error occurred.

Internal variable to record last error

Referenced by hpdftbl_get_last_err_file().

16.8.6.6 hpdftbl_err_row

`int hpdftbl_err_row  [extern]`

The row where the last error was generated.

Internal variable to record last error

16.9 hpdftbl.h

Go to the documentation of this file.
```
1
31 #include "config.h"
32 #ifndef hpdftbl_H
33 #define    hpdftbl_H
34
35 #ifdef    __cplusplus
36 // in case we have C++ code, we should use its' types and logic
37 #include <algorithm>
38 typedef std::_Bool _Bool;
39 #endif
40
41 #ifdef    __cplusplus
42 extern "C" {
43 #endif
44
45 #ifndef TRUE
47 #define TRUE 1
48 #endif
49
50 #ifndef FALSE
52 #define FALSE 0
53 #endif
54
55 #ifndef max
56
60 #define max(a,b)  (((a)>(b)) ? (a):(b))
61
65 #define min(a,b)  (((a)<(b)) ? (a):(b))
66 #endif
67
69 extern int hpdftbl_err_code ;
70
72 extern int hpdftbl_err_row ;
73
75 extern int hpdftbl_err_col ;
76
78 extern int hpdftbl_err_lineno;
79
81 extern char *hpdftbl_err_file;
82
84 extern char hpdftbl_err_extrainfo[];
85
87 #define THEME_JSON_VERSION 1
88
90 #define TABLE_JSON_VERSION 1
91
92
94 #define HPDF_FF_TIMES "Times-Roman"
96 #define HPDF_FF_TIMES_ITALIC "Times-Italic"
98 #define HPDF_FF_TIMES_BOLD "Times-Bold"
100 #define HPDF_FF_TIMES_BOLDITALIC "Times-BoldItalic"
102 #define HPDF_FF_HELVETICA "Helvetica"
104 #define HPDF_FF_HELVETICA_ITALIC "Helvetica-Oblique"
106 #define HPDF_FF_HELVETICA_BOLD "Helvetica-Bold"
108 #define HPDF_FF_HELVETICA_BOLDITALIC "Helvetica-BoldOblique"
110 #define HPDF_FF_COURIER "Courier"
112 #define HPDF_FF_COURIER_BOLD "Courier-Bold"
114 #define HPDF_FF_COURIER_IALIC "Courier-Oblique"
116 #define HPDF_FF_COURIER_BOLDITALIC "Courier-BoldOblique"
117
118
119
120 #ifdef __cplusplus
124 #define _TO_HPDF_RGB(r, g, b) \
125     { r / 255.0f, g / 255.0f, b / 255.0f }
126 #else
130 #define HPDF_RGB_CONVERT(r, g, b) \
131     (HPDF_RGBColor) { r / 255.0f, g / 255.0f, b / 255.0f }
132 #endif
133
134 #ifdef __cplusplus
135
136 #define HPDF_COLOR_DARK_RED      { 0.6f, 0.0f, 0.0f }
137 #define HPDF_COLOR_RED           { 1.0f, 0.0f, 0.0f }
138 #define HPDF_COLOR_LIGHT_GREEN   { 0.9f, 1.0f, 0.9f }
139 #define HPDF_COLOR_GREEN         { 0.4f, 0.9f, 0.4f }
140 #define HPDF_COLOR_DARK_GREEN    { 0.05f, 0.37f, 0.02f }
141 #define HPDF_COLOR_DARK_GRAY     { 0.2f, 0.2f, 0.2f }
142 #define HPDF_COLOR_LIGHT_GRAY    { 0.9f, 0.9f, 0.9f }
143 #define HPDF_COLOR_XLIGHT_GRAY   { 0.95f, 0.95f, 0.95f }
144 #define HPDF_COLOR_GRAY          { 0.5f, 0.5f, 0.5f }
145 #define HPDF_COLOR_SILVER        { 0.75f, 0.75f, 0.75f }
```

```
146 #define HPDF_COLOR_LIGHT_BLUE    { 1.0f, 1.0f, 0.9f }
147 #define HPDF_COLOR_BLUE          { 0.0f, 0.0f, 1.0f }
148 #define HPDF_COLOR_DARK_BLUE     { 0.0f, 0.0f, 0.6f }
149 #define HPDF_COLOR_WHITE         { 1.0f, 1.0f, 1.0f }
150 #define HPDF_COLOR_BLACK         { 0.0f, 0.0f, 0.0f }
151
152 #else
153
154 #define HPDF_COLOR_DARK_RED      (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
155 #define HPDF_COLOR_RED           (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
156 #define HPDF_COLOR_LIGHT_GREEN   (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
157 #define HPDF_COLOR_GREEN         (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
158 #define HPDF_COLOR_DARK_GREEN    (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
159 #define HPDF_COLOR_DARK_GRAY     (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
160 #define HPDF_COLOR_LIGHT_GRAY    (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
161 #define HPDF_COLOR_XLIGHT_GRAY   (HPDF_RGBColor) { 0.95f, 0.95f, 0.95f }
162 #define HPDF_COLOR_GRAY          (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
163 #define HPDF_COLOR_SILVER        (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
164 #define HPDF_COLOR_LIGHT_BLUE    (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
165 #define HPDF_COLOR_BLUE          (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
166 #define HPDF_COLOR_DARK_BLUE     (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
167 #define HPDF_COLOR_WHITE         (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
168 #define HPDF_COLOR_BLACK         (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
169
170 #endif
171
172 #define HPDF_COLOR_ORANGE        HPDF_RGB_CONVERT(0xF5, 0xD0, 0x98);
173 #define HPDF_COLOR_ALMOST_BLACK  HPDF_RGB_CONVERT(0x14, 0x14, 0x14);
174
183 #define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5
184
188 #define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"
189
193 #define HPDFTBL_DEFAULT_SOURCE_ENCODING "UTF-8"
194
198 #define A4PAGE_HEIGHT_CM 29.7
199
203 #define A4PAGE_WIDTH_CM 21.0
204
208 #define A3PAGE_HEIGHT_CM 42.0
209
213 #define A3PAGE_WIDTH_CM 29.7
214
218 #define LETTERRPAGE_HEIGHT_CM 27.9
219
223 #define LETTERRPAGE_WIDTH_CM 21.6
224
228 #define LEGALPAGE_HEIGHT_CM 35.6
229
233 #define LEGALPAGE_WIDTH_CM 21.6
234
238 #define HPDFTBL_END_CELLSPECS {0, 0, 0, 0, 0, 0, 0, 0, 0}
239
243 #define HPDF_COLOR_FROMRGB(r, g, b) (HPDF_RGBColor){(r)/255.0,(g)/255.0,(b)/255.0}
244
248 #define HPDFTBL_MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0
249
256 #define hpdftbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54*72)
257
265 #define _HPDFTBL_SET_ERR(t, err, r, c) do
      {hpdftbl_err_code=err;hpdftbl_err_row=r;hpdftbl_err_col=c;hpdftbl_err_lineno=__LINE__;hpdftbl_err_file=__FILE__;
      if(hpdftbl_err_handler){hpdftbl_err_handler(t,r,c,err);}} while(0)
266
273 #define _HPDFTBL_SET_ERR_EXTRA(info) do
      {strncpy(hpdftbl_err_extrainfo,info,1023);hpdftbl_err_extrainfo[1023]=0;} while(0)
274
278 #define _HPDFTBL_CHK_TABLE(t) do {if(NULL == t)
      {hpdftbl_err_code=-3;hpdftbl_err_row=-1;hpdftbl_err_col=-1;return -1;}} while(0)
279
283 #define _HPDFTBL_IDX(r, c) (r*t->cols+c)
284
292 typedef enum hpdftbl_text_align {
293     LEFT = 0,
294     CENTER = 1,
295     RIGHT = 2
296 } hpdftbl_text_align_t;
297
303 typedef struct text_style {
304     char *font;
305     HPDF_REAL fsize;
306     HPDF_RGBColor color;
307     HPDF_RGBColor background;
308     hpdftbl_text_align_t halign;
309 } hpdf_text_style_t;
310
311
320 typedef struct hpdftbl *hpdftbl_t;
```

```
321
330 typedef char *(*hpdftbl_content_callback_t)(void *, size_t, size_t);
331
341 typedef void (*hpdftbl_canvas_callback_t)(HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_REAL,
         HPDF_REAL, HPDF_REAL,
342                                             HPDF_REAL);
343
353 typedef _Bool (*hpdftbl_content_style_callback_t)(void *, size_t, size_t, char *content,
         hpdf_text_style_t *);
354
355
365 typedef void (*hpdftbl_callback_t)(hpdftbl_t);
366
375 typedef enum hpdftbl_dashstyle {
376     LINE_SOLID ,
377     LINE_DOT1 ,
378     LINE_DOT2 ,
379     LINE_DOT3 ,
380     LINE_DOT4 ,
381     LINE_DASH1 ,
382     LINE_DASH2 ,
383     LINE_DASH3 ,
384     LINE_DASH4 ,
385     LINE_DASH5 ,
386     LINE_DASHDOT1 ,
387     LINE_DASHDOT2
388 } hpdftbl_line_dashstyle_t;
389
395 typedef struct grid_style {
396     HPDF_REAL width;
397     HPDF_RGBColor color;
398     hpdftbl_line_dashstyle_t line_dashstyle;
399 } hpdftbl_grid_style_t;
400
408 struct hpdftbl_cell {
410     size_t row;
412     size_t col;
414     char *label;
416     char *content;
418     size_t colspan;
420     size_t rowspan;
422     HPDF_REAL height;
424     HPDF_REAL width;
426     HPDF_REAL delta_x;
428     HPDF_REAL delta_y;
430     HPDF_REAL textwidth;
432     hpdftbl_content_callback_t content_cb;
434     char *content_dyncb;
436     hpdftbl_content_callback_t label_cb;
438     char *label_dyncb;
440     hpdftbl_content_style_callback_t style_cb;
442     char *content_style_dyncb;
444     hpdftbl_canvas_callback_t canvas_cb;
446     char *canvas_dyncb;
448     hpdf_text_style_t content_style;
452     struct hpdftbl_cell *parent_cell;
453 };
454
460 typedef struct hpdftbl_cell hpdftbl_cell_t;
461
470 struct hpdftbl {
472     HPDF_Doc pdf_doc;
474     HPDF_Page pdf_page;
476     size_t cols;
478     size_t rows;
480     HPDF_REAL posx;
482     HPDF_REAL posy;
484     HPDF_REAL height;
486     HPDF_REAL minrowheight;
488     _Bool anchor_is_top_left;
490     HPDF_REAL bottom_vmargin_factor;
492     HPDF_REAL width;
494     void *tag;
496     char *title_txt;
498     hpdf_text_style_t title_style;
500     hpdf_text_style_t header_style;
502     _Bool use_header_row;
504     hpdf_text_style_t label_style;
506     _Bool use_cell_labels;
508     _Bool use_label_grid_style;
510     hpdf_text_style_t content_style;
512     hpdftbl_content_callback_t label_cb;
514     char *label_dyncb;
516     hpdftbl_content_callback_t content_cb;
518     char *content_dyncb;
520     hpdftbl_content_style_callback_t content_style_cb;
522     char *content_style_dyncb;
```

```
524    hpdftbl_canvas_callback_t canvas_cb;
526    char *canvas_dyncb;
531    hpdftbl_callback_t post_cb;
533    char *post_dyncb;
535    hpdftbl_grid_style_t outer_grid;
537    hpdftbl_grid_style_t inner_vgrid;
539    hpdftbl_grid_style_t inner_hgrid;
541    hpdftbl_grid_style_t inner_tgrid;
545    _Bool use_zebra;
549    int zebra_phase;
551    HPDF_RGBColor zebra_color1;
553    HPDF_RGBColor zebra_color2;
555    float *col_width_percent;
557    hpdftbl_cell_t *cells;
558 };
559
569 typedef struct hpdftbl_cell_spec {
571    size_t row;
573    size_t col;
575    unsigned rowspan;
577    unsigned colspan;
579    char *label;
581    hpdftbl_content_callback_t content_cb;
583    hpdftbl_content_callback_t label_cb;
585    hpdftbl_content_style_callback_t style_cb;
587    hpdftbl_canvas_callback_t canvas_cb;
588 } hpdftbl_cell_spec_t;
589
596 typedef struct hpdftbl_spec {
598    char *title;
600    _Bool use_header;
602    _Bool use_labels;
604    _Bool use_labelgrid;
606    size_t rows;
608    size_t cols;
610    HPDF_REAL xpos;
612    HPDF_REAL ypos;
614    HPDF_REAL width;
616    HPDF_REAL height;
618    hpdftbl_content_callback_t content_cb;
620    hpdftbl_content_callback_t label_cb;
622    hpdftbl_content_style_callback_t style_cb;
624    hpdftbl_callback_t post_cb;
626    hpdftbl_cell_spec_t *cell_spec;
627 } hpdftbl_spec_t;
628
635 typedef struct hpdftbl_theme {
637    hpdf_text_style_t content_style;
639    hpdf_text_style_t label_style;
641    hpdf_text_style_t header_style;
643    hpdf_text_style_t title_style;
645    hpdftbl_grid_style_t outer_border;
647    _Bool use_labels;
649    _Bool use_label_grid_style;
651    _Bool use_header_row;
653    hpdftbl_grid_style_t inner_vborder;
655    hpdftbl_grid_style_t inner_hborder;
657    hpdftbl_grid_style_t inner_tborder;
659    _Bool use_zebra;
661    int zebra_phase;
663    HPDF_RGBColor zebra_color1;
665    HPDF_RGBColor zebra_color2;
667    HPDF_REAL bottom_vmargin_factor;
668 } hpdftbl_theme_t;
669
677 typedef void (*hpdftbl_error_handler_t)(hpdftbl_t, int, int, int);
678
679 extern hpdftbl_error_handler_t hpdftbl_err_handler ;
680
681 /*
682  * Table creation and destruction function
683  */
684 hpdftbl_t
685 hpdftbl_create(size_t rows, size_t cols);
686
687 hpdftbl_t
688 hpdftbl_create_title(size_t rows, size_t cols, char *title);
689
690 int
691 hpdftbl_stroke(HPDF_Doc pdf,
692                HPDF_Page page, hpdftbl_t t,
693                HPDF_REAL xpos, HPDF_REAL ypos,
694                HPDF_REAL width, HPDF_REAL height);
695
696 int
697 hpdftbl_stroke_pos(HPDF_Doc pdf,
698                    const HPDF_Page page, hpdftbl_t t);
```

```
699
700 int
701 hpdftbl_stroke_from_data(HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t *tbl_spec, hpdftbl_theme_t
      *theme);
702
703 int
704 hpdftbl_setpos(hpdftbl_t t,
705                const HPDF_REAL xpos, const HPDF_REAL ypos,
706                const HPDF_REAL width, HPDF_REAL height);
707
708 int
709 hpdftbl_destroy(hpdftbl_t t);
710
711 int
712 hpdftbl_get_last_auto_height(HPDF_REAL *height);
713
714 void
715 hpdftbl_set_anchor_top_left(hpdftbl_t tbl, _Bool anchor);
716
717 _Bool
718 hpdftbl_get_anchor_top_left(hpdftbl_t tbl);
719
720 /*
721  * Table error handling functions
722  */
723 hpdftbl_error_handler_t
724 hpdftbl_set_errhandler(hpdftbl_error_handler_t);
725
726 const char *
727 hpdftbl_get_errstr(int err);
728
729 const char *
730 hpdftbl_hpdf_get_errstr(HPDF_STATUS err_code);
731
732 int
733 hpdftbl_get_last_errcode(const char **errstr, int *row, int *col);
734
735 void
736 hpdftbl_get_last_err_file(int *lineno, char **file, char **extrainfo);
737
738 void
739 hpdftbl_default_table_error_handler(hpdftbl_t t, int r, int c, int err);
740
741 /*
742  * Theme handling functions
743  */
744 int
745 hpdftbl_apply_theme(hpdftbl_t t, hpdftbl_theme_t *theme);
746
747 hpdftbl_theme_t *
748 hpdftbl_get_default_theme(void);
749
750 int
751 hpdftbl_get_theme(hpdftbl_t tbl, hpdftbl_theme_t *theme);
752
753 int
754 hpdftbl_destroy_theme(hpdftbl_theme_t *theme);
755
756 /*
757  * Table layout adjusting functions
758  */
759
760 void
761 hpdftbl_set_bottom_vmargin_factor(hpdftbl_t t, HPDF_REAL f);
762
763 int
764 hpdftbl_set_min_rowheight(hpdftbl_t t, float h);
765
766 int
767 hpdftbl_set_colwidth_percent(hpdftbl_t t, size_t c, float w);
768
769 int
770 hpdftbl_clear_spanning(hpdftbl_t t);
771
772 int
773 hpdftbl_set_cellspan(hpdftbl_t t, size_t r, size_t c, size_t rowspan, size_t colspan);
774
775 /*
776  * Table style handling functions
777  */
778 int
779 hpdftbl_set_zebra(hpdftbl_t t, _Bool use, int phase);
780
781 int
782 hpdftbl_set_zebra_color(hpdftbl_t t, HPDF_RGBColor z1,  HPDF_RGBColor z2);
783
784 int
```

```
785 hpdftbl_use_labels(hpdftbl_t t, _Bool use);
786
787 int
788 hpdftbl_use_labelgrid(hpdftbl_t t, _Bool use);
789
790 int
791 hpdftbl_set_background(hpdftbl_t t, HPDF_RGBColor background);
792
793 int
794 hpdftbl_set_inner_tgrid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color,
795     hpdftbl_line_dashstyle_t dashstyle);
795
796 int
797 hpdftbl_set_inner_vgrid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color,
798     hpdftbl_line_dashstyle_t dashstyle);
798
799 int
800 hpdftbl_set_inner_hgrid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color,
801     hpdftbl_line_dashstyle_t dashstyle);
801
802 int
803 hpdftbl_set_inner_grid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t
804     dashstyle);
804
805 int
806 hpdftbl_set_outer_grid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t
807     dashstyle);
807
808 int
809 hpdftbl_set_header_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
810     background);
810
811 int
812 hpdftbl_set_header_halign(hpdftbl_t t, hpdftbl_text_align_t align);
813
814 int
815 hpdftbl_use_header(hpdftbl_t t, _Bool use);
816
817 int
818 hpdftbl_set_label_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
819     background);
819
820 int
821 hpdftbl_set_row_content_style(hpdftbl_t t, size_t r, char *font, HPDF_REAL fsize, HPDF_RGBColor color,
822                               HPDF_RGBColor background);
823
824 int
825 hpdftbl_set_col_content_style(hpdftbl_t t, size_t c, char *font, HPDF_REAL fsize, HPDF_RGBColor color,
826                               HPDF_RGBColor background);
827
828 int
829 hpdftbl_set_content_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
830     background);
831 int
832 hpdftbl_set_cell_content_style(hpdftbl_t t, size_t r, size_t c, char *font, HPDF_REAL fsize,
833     HPDF_RGBColor color,
833                                HPDF_RGBColor background);
834
835 int
836 hpdftbl_set_title_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
837     background);
837
838 /*
839  * Table content handling
840  */
841 int
842 hpdftbl_set_cell(hpdftbl_t t, size_t r, size_t c, char *label, char *content);
843
844 int
845 hpdftbl_set_tag(hpdftbl_t t, void *tag);
846
847 int
848 hpdftbl_set_title(hpdftbl_t t, char *title);
849
850 int
851 hpdftbl_set_title_halign(hpdftbl_t t, hpdftbl_text_align_t align);
852
853 int
854 hpdftbl_set_labels(hpdftbl_t t, char **labels);
855
856 int
857 hpdftbl_set_content(hpdftbl_t t, char **content);
858
859 /*
860  * Table callback functions
861  */
```

```
862 int
863 hpdftbl_set_content_cb(hpdftbl_t t, hpdftbl_content_callback_t cb);
864
865 int
866 hpdftbl_set_cell_content_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb);
867
868 int
869 hpdftbl_set_label_cb(hpdftbl_t t, hpdftbl_content_callback_t cb);
870
871 int
872 hpdftbl_set_cell_label_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb);
873
874 int
875 hpdftbl_set_canvas_cb(hpdftbl_t t, hpdftbl_canvas_callback_t cb);
876
877 int
878 hpdftbl_set_cell_canvas_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_canvas_callback_t cb);
879
880 int
881 hpdftbl_set_content_style_cb(hpdftbl_t t, hpdftbl_content_style_callback_t cb);
882
883 int
884 hpdftbl_set_cell_content_style_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_style_callback_t cb);
885
886 int
887 hpdftbl_set_post_cb(hpdftbl_t t, hpdftbl_callback_t cb);
888
889 /*
890  * Table dynamic callback functions
891  */
892 void
893 hpdftbl_set_dlhandle(void *);
894
895 int
896 hpdftbl_set_content_dyncb(hpdftbl_t, const char *);
897
898 int
899 hpdftbl_set_canvas_dyncb(hpdftbl_t, const char *);
900
901 int
902 hpdftbl_set_cell_content_dyncb(hpdftbl_t, size_t, size_t, const char *);
903
904 int
905 hpdftbl_set_label_dyncb(hpdftbl_t, const char *);
906
907 int
908 hpdftbl_set_cell_label_dyncb(hpdftbl_t, size_t, size_t, const char *);
909
910 int
911 hpdftbl_set_content_style_dyncb(hpdftbl_t, const char *);
912
913 int
914 hpdftbl_set_cell_content_style_dyncb(hpdftbl_t, size_t, size_t, const char *);
915
916 int
917 hpdftbl_set_cell_canvas_dyncb(hpdftbl_t, size_t, size_t, const char *);
918
919 int
920 hpdftbl_set_post_dyncb(hpdftbl_t t, const char *cb_name);
921
922 /*
923  * Text encoding
924  */
925 void
926 hpdftbl_set_text_encoding(char *target, char *source);
927
928 int
929 hpdftbl_encoding_text_out(HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, char *text);
930
931 /*
932  * Misc utility and widget functions
933  */
934
935 void
936 HPDF_RoundedCornerRectangle(HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL
    height,
937                             HPDF_REAL rad);
938
939 void
940 hpdftbl_stroke_grid(HPDF_Doc pdf, HPDF_Page page);
941
942 void
943 hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
944                                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
945                                     HPDF_RGBColor on_color, HPDF_RGBColor off_color,
946                                     HPDF_RGBColor on_background, HPDF_RGBColor off_background,
947                                     HPDF_REAL fsize,
```

```
948                                          const char *letters, _Bool *state);
949
950 void
951 hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
952                             HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, _Bool
       state);
953
954 void
955 hpdftbl_widget_hbar(HPDF_Doc doc, HPDF_Page page,
956                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
957                     HPDF_RGBColor color, float val, _Bool hide_val);
958
959 void
960 hpdftbl_widget_segment_hbar(HPDF_Doc doc, HPDF_Page page,
961                             HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
962                             size_t num_segments, HPDF_RGBColor on_color, double val_percent,
963                             _Bool hide_val);
964
965 void
966 hpdftbl_widget_strength_meter(HPDF_Doc doc, HPDF_Page page,
967                               HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
968                               size_t num_segments, HPDF_RGBColor on_color, size_t num_on_segments);
969
970 int
971 hpdftbl_stroke_pdfdoc(HPDF_Doc pdf_doc, char *file);
972
973 #ifdef HAVE_LIBJANSSON
974
975 int
976 hpdftbl_dump(hpdftbl_t tbl, char *filename);
977
978 int
979 hpdftbl_dumps(hpdftbl_t tbl, char *buff, size_t buffsize);
980
981 int
982 hpdftbl_load(hpdftbl_t tbl, char *filename);
983
984 int
985 hpdftbl_loads(hpdftbl_t tbl, char *buff);
986
987 int
988 hpdftbl_theme_dump(hpdftbl_theme_t *theme, char *filename);
989
990 int
991 hpdftbl_theme_dumps(hpdftbl_theme_t *theme, char *buff, size_t buffsize);
992
993 int
994 hpdftbl_theme_loads(hpdftbl_theme_t *tbl, char *buff);
995
996 int
997 hpdftbl_theme_load(hpdftbl_theme_t *tbl, char *filename);
998
999 #endif
1000
1001 size_t
1002 xstrlcat(char *dst, const char *src, size_t siz);
1003
1004 size_t
1005 xstrlcpy(char * __restrict dst, const char * __restrict src, size_t dsize);
1006
1007 int
1008 hpdftbl_read_file(char *buff, size_t buffsize, char *filename);
1009
1010 /*
1011  * Internal functions
1012  */
1013 _Bool
1014 chktbl(hpdftbl_t, size_t, size_t);
1015
1016 #ifdef    __cplusplus
1017 }
1018 #endif
1019
1020 #endif    /* hpdftbl_H */
```

16.10 hpdftbl_callback.c File Reference

Routines for plain and dynamic callback function.

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

```
#include <unistd.h>
#include <string.h>
#include <iconv.h>
#include <hpdf.h>
#include <libgen.h>
#include <sys/stat.h>
#include <dlfcn.h>
#include "hpdftbl.h"
```

Functions

- void hpdftbl_set_dlhandle (void ∗handle)

  Set the handle for scope of dynamic function search.
- int hpdftbl_set_content_cb (hpdftbl_t t, hpdftbl_content_callback_t cb)

  Set table content callback.
- int hpdftbl_set_cell_content_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb)

  Set cell content callback.
- int hpdftbl_set_cell_label_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb)

  Set cell label callback.
- int hpdftbl_set_cell_canvas_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_canvas_callback_t cb)

  Set cell canvas callback.
- int hpdftbl_set_label_cb (hpdftbl_t t, hpdftbl_content_callback_t cb)

  Set table label callback.
- int hpdftbl_set_post_cb (hpdftbl_t t, hpdftbl_callback_t cb)

  Set table post processing callback.
- int hpdftbl_set_canvas_cb (hpdftbl_t t, hpdftbl_canvas_callback_t cb)

  Set cell canvas callback.
- int hpdftbl_set_content_dyncb (hpdftbl_t t, const char ∗cb_name)

  Specify dynamic (late) loading callback content function.
- int hpdftbl_set_canvas_dyncb (hpdftbl_t t, const char ∗cb_name)

  Specify dynamic (late) loading callback content function.
- int hpdftbl_set_label_dyncb (hpdftbl_t t, const char ∗cb_name)

  Specify dynamic (late) loading callback for table label function.
- int hpdftbl_set_cell_label_dyncb (hpdftbl_t t, size_t r, size_t c, const char ∗cb_name)

  Specify dynamic (late) loading callback for cell label function.
- int hpdftbl_set_content_style_dyncb (hpdftbl_t t, const char ∗cb_name)

  Specify dynamic (late) loading callback for table style function.
- int hpdftbl_set_cell_content_style_dyncb (hpdftbl_t t, size_t r, size_t c, const char ∗cb_name)

  Specify dynamic (late) loading callback for cell style function.
- int hpdftbl_set_cell_content_dyncb (hpdftbl_t t, size_t r, size_t c, const char ∗cb_name)

  Specify dynamic (late) loading callback cell content function.
- int hpdftbl_set_cell_canvas_dyncb (hpdftbl_t t, size_t r, size_t c, const char ∗cb_name)

  Specify dynamic (late) loading callback cell canvas function.
- int hpdftbl_set_post_dyncb (hpdftbl_t t, const char ∗cb_name)

  Set table post processing callback.
- int hpdftbl_set_cell_content_style_cb (hpdftbl_t t, size_t r, size_t c, hpdftbl_content_style_callback_t cb)

  Set cell specific callback to specify cell content style.
- int hpdftbl_set_content_style_cb (hpdftbl_t t, hpdftbl_content_style_callback_t cb)

  Set callback to specify cell content style.

16.10.1 Detailed Description

Routines for plain and dynamic callback function.

All functions ending with `_cb` are used to specify standard callback functions which stores a function pointer bounded at compile time. All functions ending i `_dyncb` are used to set dynamic callback functions which are bound at run time. The function name are stored as string and resolved at runtime.

Author

Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

LICENSE

16.10.2 Function Documentation

16.10.2.1 hpdftbl_set_canvas_cb()

```
int hpdftbl_set_canvas_cb (
            hpdftbl_t t,
            hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set cell canvas callback. This callback gets called for each cell in the table. The purpose is to allow the client to add dynamic content to the specified cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell. To set the canvas callback only for a specific cell use the hpdftbl_set_cell_canvas_cb() function

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_set_cell_canvas_cb()

16.10.2.2 hpdftbl_set_canvas_dyncb()

```
int hpdftbl_set_canvas_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as canvas callback. This function must follow the signature of a callback function as specified in hpdftbl_canvas_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_canvas_cb(), hpdftbl_canvas_callback_t

16.10.2.3 hpdftbl_set_cell_canvas_cb()

```
int hpdftbl_set_cell_canvas_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set a canvas callback for an individual cell. This will override the table canvas callback. The canvas callback is called with arguments that give the bounding box for the cell. In that way a callback function may draw arbitrary graphic in the cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

[hpdftbl_canvas_callback_t](#)

[hpdftbl_set_canvas_cb()](#)

Referenced by [hpdftbl_stroke_from_data()](#).

16.10.2.4   hpdftbl_set_cell_canvas_dyncb()

```
int hpdftbl_set_cell_canvas_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback cell canvas function.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file()](#) to read it back.

Parameters

t	Table handle
r	Celll row
c	Cell column
cb_name	Name of the function to be used as canvas callback. This function must follow the signature of a callback function as specified in hpdftbl_canvas_callback_t

Returns

See also

[hpdftbl_set_cell_canvas_cb()](#), [hpdftbl_canvas_callback_t](#)

16.10.2.5   hpdftbl_set_cell_content_cb()

```
int hpdftbl_set_cell_content_cb (
            hpdftbl_t t,
            size_t r,
```

```
            size_t c,
            hpdftbl_content_callback_t cb )
```

Set cell content callback.

Set a content callback for an individual cell. This will override the table content callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

t	Table handle
cb	Callback function
r	Cell row
c	Cell column

Returns

-1 on failure, 0 otherwise

See also

[hpdftbl_set_content_cb()](#)

Referenced by [hpdftbl_stroke_from_data()](#).

16.10.2.6 hpdftbl_set_cell_content_dyncb()

```
int hpdftbl_set_cell_content_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback cell content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file()](#) to read it back.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb_name	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_content_cb()](#), [hpdftbl_content_callback_t](#)

16.10.2.7 hpdftbl_set_cell_content_style_cb()

```
int hpdftbl_set_cell_content_style_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_content_style_callback_t cb )
```

Set cell specific callback to specify cell content style.

Set callback to format the style for the specified cell

Parameters

t	Table handle
r	Cell row
c	Cell column
cb	Callback function

Returns

0 on success, -1 on failure

See also

hpdftbl_set_ontent_style_cb()

Referenced by hpdftbl_stroke_from_data().

16.10.2.8 hpdftbl_set_cell_content_style_dyncb()

```
int hpdftbl_set_cell_content_style_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback for cell style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by hpdftbl_content_style_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_style_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_cell_content_style_cb(), hpdftbl_content_style_callback_t

16.10.2.9 hpdftbl_set_cell_label_cb()

```
int hpdftbl_set_cell_label_cb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            hpdftbl_content_callback_t cb )
```

Set cell label callback.

Set a label callback for an individual cell. This will override the table label callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

t	Table handle
cb	Callback function
r	Cell row
c	Cell column

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_set_label_cb()

Referenced by hpdftbl_stroke_from_data().

16.10.2.10 hpdftbl_set_cell_label_dyncb()

```
int hpdftbl_set_cell_label_dyncb (
            hpdftbl_t t,
            size_t r,
            size_t c,
            const char * cb_name )
```

Specify dynamic (late) loading callback for cell label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by hpdftbl_content_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
r	Cell row
c	Cell column
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_cell_label_cb(), hpdftbl_content_callback_t

16.10.2.11 hpdftbl_set_content_cb()

```
int hpdftbl_set_content_cb (
            hpdftbl_t t,
            hpdftbl_content_callback_t cb )
```

Set table content callback.

This callback gets called for each cell in the table and the returned string will be used as the content. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the content will be set to the content specified with the direct content setting. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

t	Table handle
cb	Callback function

Returns

-1 for error , 0 otherwise

See also

hpdftbl_set_cell_content_cb()

Referenced by hpdftbl_stroke_from_data().

16.10.2.12 hpdftbl_set_content_dyncb()

```
int hpdftbl_set_content_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_content_cb(), hpdftbl_content_callback_t

16.10.2.13 hpdftbl_set_content_style_cb()

```
int hpdftbl_set_content_style_cb (
            hpdftbl_t t,
            hpdftbl_content_style_callback_t cb )
```

Set callback to specify cell content style.

Set callback to format the style for cells in the table. If a cell has its own content style callback that callback will override the generic table callback.

Parameters

t	Table handle
cb	Callback function

Returns

0 on success, -1 on failure

See also

[hpdftbl_set_cell_content_style_cb()](#)

Referenced by [hpdftbl_stroke_from_data()](#).

16.10.2.14 hpdftbl_set_content_style_dyncb()

```
int hpdftbl_set_content_style_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback for table style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by hpdftbl_content_style_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file()](#) to read it back.

Parameters

_t_	Table handle
_cb_name_	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_style_callback_t.

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_content_style_cb()](#), [hpdftbl_content_style_callback_t](#)

16.10.2.15 hpdftbl_set_dlhandle()

```
void hpdftbl_set_dlhandle (
            void * handle )
```

Set the handle for scope of dynamic function search.

When using late binding (some os _dyncb() functions) the scope for where the runtime searches for the functions can be specified as is discussed in `man 3 dlsym`. By default the library uses dl_handle which make the library first searches the current image and then all images it was built against.

If the dynamic callbacks are located in a runtime loaded library then the handle returned by dlopen() must be specified as the function will not be found otherwise.

Parameters

handle	Predefined values or the handle returned by dlopen() (see man dlopen)

16.10.2.16 hpdftbl_set_label_cb()

```
int hpdftbl_set_label_cb (
            hpdftbl_t t,
            hpdftbl_content_callback_t cb )
```

Set table label callback.

Set label callback. This callback gets called for each cell in the table and the returned string will be used as the label. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the label will be set to the content specified with the direct label setting. The callback function will receive the Table tag and the row and column

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_content_callback_t

hpdftbl_set_cell_label_cb()

Referenced by hpdftbl_stroke_from_data().

16.10.2.17 hpdftbl_set_label_dyncb()

```
int hpdftbl_set_label_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Specify dynamic (late) loading callback for table label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. symbols. The callback function must have the signature defined by hpdftbl_content_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the hpdftbl_get_last_err_file() to read it back.

Parameters

t	Table handle
cb_name	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in hpdftbl_content_callback_t.

Returns

-1 on failure, 0 on success

See also

hpdftbl_set_label_cb(), hpdftbl_content_callback_t

16.10.2.18 hpdftbl_set_post_cb()

```
int hpdftbl_set_post_cb (
            hpdftbl_t t,
            hpdftbl_callback_t cb )
```

Set table post processing callback.

This is an optional post processing callback for anything in general to do after the table has been constructed. The callback happens after the table has been fully constructed and just before it is stroked.

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_callback_t

16.10.2.19 hpdftbl_set_post_dyncb()

```
int hpdftbl_set_post_dyncb (
            hpdftbl_t t,
            const char * cb_name )
```

Set table post processing callback.

This is an optional post processing callback for anything in general to do after the table has been constructed. The callback only gets the table as its first and only argument. The callback happens after the table has been fully constructed and just before it is stroked.

Parameters

t	Table handle
cb_name	Callback function name

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_callback_t, hpdftbl_set_post_cb()

16.11 hpdftbl_dump.c File Reference

Functions for json serializing of table data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <hpdf.h>
#include "hpdftbl.h"
#include <jansson.h>
```

Macros

- #define **jsonprint**(...) if(_string_) { sprintf (_sbuff_, __VA_ARGS__);size_t _len_=xstrlcat(_jsonbuff_,_↵ sbuff_,_jsonbuff_size_); if(_len_>=_jsonbuff_size_) return -1;} else {fprintf(fh,__VA_ARGS__);}
- #define **OUTJSON_NEWBLK**() do { jsonprint("%∗s{\n", tab,""); } while(0)
- #define **OUTJSON_ENDDOC**() do { jsonprint("}\n"); } while(0)
- #define **OUTJSON_INDENT**() do { jsonprint("%∗s", tab,""); } while(0)
- #define **OUTJSON_NEWLINE**() do { jsonprint("\n"); } while(0)
- #define **OUTJSON_STRINT**(k, v, e) do { jsonprint("%s\"%s\": %d%c\n",tab,"",k,(int)v,e); } while(0)
- #define **OUTJSON_STRREAL**(k, v, e) do { jsonprint("%s\"%s\": %.8f%c\n",tab,"",k,v,e); } while(0)
- #define **OUTJSON_STRSTR**(k, v) do { if(v==NULL) {jsonprint("%∗s\"%s\": \"\",\n",tab,"",k) } else { jsonprint("%∗s\"%s\": \"%s\",\n",tab,"",k,v);} } while(0)
- #define **OUTJSON_STRBLK**(k) do { jsonprint("%∗s\"%s\": {\n",tab,"",k); } while(0)
- #define **OUTJSON_STRLIST**(k) do { jsonprint("%∗s\"%s\": [\n",tab,"",k); } while(0)
- #define **OUTJSON_ENDLIST**(e) do { jsonprint("\n%∗s]%c\n",tab,"",e); } while(0)
- #define **OUTJSON_STARTBLK**() do { jsonprint("%∗s{\n",tab,""); } while(0)
- #define **OUTJSON_ENDBLK**(c) do { jsonprint("%∗s}%c\n",tab,"",c); } while(0)
- #define **OUTJSON_BOOL**(k, v) do { if(v) { jsonprint("%∗s\"%s\": true,\n",tab,"",k);} else { jsonprint("%∗s\"%s\": false,\n",tab,"",k);} } while(0)
- #define **OUTJSON_RGB**(k, v) do { jsonprint("%∗s\"%s\": [%.5f, %.5f, %.5f],\n",tab,"",k,v.r,v.g,v.b); } while(0)
- #define OUTJSON_GRID(k, v)
- #define OUTJSON_TXTSTYLE(k, v, c)

Functions

- int hpdftbl_theme_dump (hpdftbl_theme_t ∗theme, char ∗filename)

    Serialize the specified theme structure to a named file.

- int hpdftbl_theme_dumps (hpdftbl_theme_t ∗theme, char ∗buff, const size_t buffsize)

    Serialize theme structure to a string buffer.

- int hpdftbl_dump (hpdftbl_t tbl, char ∗filename)

    Serialize a table structure as a JSON file.

- int hpdftbl_dumps (hpdftbl_t tbl, char ∗buff, size_t buffsize)

    Serialize a table structure to a string buffer.

16.11.1 Detailed Description

Functions for json serializing of table data structure.

Author

Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

LICENSE

Released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN-CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.11.2 Macro Definition Documentation

16.11.2.1 OUTJSON_GRID

```
#define OUTJSON_GRID(
                k,
                v )
```

Value:
```
    do { \
    OUTJSON_STRBLK(k);                      \
    tab += 2;                               \
    OUTJSON_STRREAL("width",v.width,',');       \
    OUTJSON_RGB("color",v.color); \
    OUTJSON_STRINT("dashstyle",v.line_dashstyle,' '); \
    tab -= 2;                               \
    OUTJSON_ENDBLK(',');                    \
} while(0)
```

16.11.2.2 OUTJSON_TXTSTYLE

```
#define OUTJSON_TXTSTYLE(
                k,
                v,
                c )
```

Value:
```
    do { \
    jsonprint( "%*s\"%s\": {\n",tab,"",k); \
    tab += 2;\
    OUTJSON_STRSTR("font",v.font); \
    OUTJSON_STRREAL("fsize",v.fsize,',');\
    OUTJSON_RGB("color",v.color);           \
    OUTJSON_RGB("background",v.background); \
    OUTJSON_STRINT("halign",v.halign,' '); \
    tab -= 2;   \
    OUTJSON_ENDBLK(c); \
} while(0)
```

16.11.3 Function Documentation

16.11.3.1 hpdftbl_dump()

```
int hpdftbl_dump (
                hpdftbl_t tbl,
                char * filename )
```

Serialize a table structure as a JSON file.

The table is serialized as JSON file and have whitespaces and newlines to make it more human readable. The serialization is a complete representation of a table.

Parameters

tbl	Table handle
filename	Filename to write to. Any path specified must exists

Returns

-1 on failure, 0 on success

16.11.3.2 hpdftbl_dumps()

```
int hpdftbl_dumps (
            hpdftbl_t tbl,
            char * buff,
            size_t buffsize )
```

Serialize a table structure to a string buffer.

The table is serialized as JSON and have whitespaces and newlines to make it more human readable. Note is is the callers responsibility to make sure the buffer is large enough to hold the serialized table.

Parameters

tbl	Table handle of table to dump
buff	Buffer to dump structure to
buffsize	Size of buffer

Returns

-1 on failure, 0 on success

See also

hpdftbl_load(),hpdftbl_dump(), hpdftbl_stroke_pos(),

Referenced by hpdftbl_dump().

16.11.3.3 hpdftbl_theme_dump()

```
int hpdftbl_theme_dump (
            hpdftbl_theme_t * theme,
            char * filename )
```

Serialize the specified theme structure to a named file.

The theme is serialized as JSON string array and have whitespaces and newlines to make it more human readable.

Parameters

theme	Pointer to theme structure to be serialized
filename	Filename to write to

Returns

0 on success, -1 on failure

16.11.3.4 hpdftbl_theme_dumps()

```
int hpdftbl_theme_dumps (
            hpdftbl_theme_t * theme,
            char * buff,
            const size_t buffsize )
```

Serialize theme structure to a string buffer.

The theme is serialized as JSON string array and have whitespaces and newlines to make it more human readable.

Parameters

theme	Theme to serialize
buff	Buffer to write serialized theme to. It should be a minimum of 2k chars.
buffsize	Buffer size (including ending string NULL)

Returns

0 on success, < 0 on failure

Referenced by hpdftbl_theme_dump().

16.12 hpdftbl_errstr.c File Reference

Utility module to translate HPDF error codes to human readable strings.

```
#include <stdio.h>
#include <hpdf.h>
#include "hpdftbl.h"
```

Data Structures

- struct hpdftbl_errcode_entry

    An entry in the error string table.

Macros

- #define **ERR_UNKNOWN** 11

    Error code for unknown error.

Functions

- const char ∗ hpdftbl_hpdf_get_errstr (const HPDF_STATUS err_code)

  Function to return a human readable error string for an error code from Core HPDF library.

- const char ∗ hpdftbl_get_errstr (int err)

  Translate a table error code to a human readable string.

- void hpdftbl_default_table_error_handler (hpdftbl_t t, int r, int c, int err)

  A basic default table error handler.

- int hpdftbl_get_last_errcode (const char ∗∗errstr, int ∗row, int ∗col)

  Return last error code.

- void hpdftbl_get_last_err_file (int ∗lineno, char ∗∗file, char ∗∗extrainfo)

  Get the filename and line number where the last error occurred.

- hpdftbl_error_handler_t hpdftbl_set_errhandler (hpdftbl_error_handler_t err_handler)

  Specify errhandler for the table routines.

Variables

- int hpdftbl_err_code = 0

  Stores the last generated error code.

- int hpdftbl_err_row = -1

  The row where the last error was generated.

- int hpdftbl_err_col = -1

  The column where the last error was generated.

- int hpdftbl_err_lineno = 0

  Hold the line number of the last error occurred.

- char ∗ hpdftbl_err_file = NULL

  Hold the file name where the last error occurred.

- char hpdftbl_err_extrainfo [1024] = {0}

  Extra info that may be specified at the point of error.

16.12.1   Detailed Description

Utility module to translate HPDF error codes to human readable strings.

16.12.2   Function Documentation

16.12.2.1   hpdftbl_default_table_error_handler()

```
void hpdftbl_default_table_error_handler (
            hpdftbl_t t,
            int r,
            int c,
            int err )
```

A basic default table error handler.

This error handler is used as a callback that outputs the error to stderr in human readable format and quits the process.

Parameters

t	Table where the error happened (can be NULL)
r	Cell row
c	Cell column
err	The error code

See also

> hpdftbl_set_errhandler()

16.12.2.2 hpdftbl_get_errstr()

```
const char * hpdftbl_get_errstr (
            int err )
```

Translate a table error code to a human readable string.

The function returns a pointer to a static string that cannot be modified. It will translate both internal table error messages as well as generic HPDF library error codes.

Parameters

err	The error code to be translated

Returns

> Static pointer to string for valid error code, NULL otherwise

See also

> hpdftbl_hpdf_get_errstr()

Referenced by hpdftbl_default_table_error_handler(), and hpdftbl_get_last_errcode().

16.12.2.3 hpdftbl_get_last_err_file()

```
void hpdftbl_get_last_err_file (
            int * lineno,
            char ** file,
            char ** extrainfo )
```

Get the filename and line number where the last error occurred.

Parameters

lineno	Set to the line number where the error occurred
file	Set to the file where the error occurred
extrainfo	Extra info string that may be set at the point of error

16.12.2.4 hpdftbl_get_last_errcode()

```
int hpdftbl_get_last_errcode (
            const char ** errstr,
            int * row,
            int * col )
```

Return last error code.

Return last error code. if errstr is not NULL a human readable string describing the error will be copied to the string. The error code will be reset after call.

Parameters

errstr	A string buffer where the error string is written to
row	The row where the error was found
col	The col where the error was found

Returns

The last error code

16.12.2.5 hpdftbl_hpdf_get_errstr()

```
const char * hpdftbl_hpdf_get_errstr (
            const HPDF_STATUS err_code )
```

Function to return a human readable error string for an error code from Core HPDF library.

The various error codes given by the HPDF library can be translated back to a string by the usage of this function. The function will return a pointer to a static string that can not be manipulated.

Parameters

err_code	The error code

Returns

A pointer to an error string, NULL if the error code is invalid

See also

    hpdftbl_get_errstr()

16.12.2.6 hpdftbl_set_errhandler()

hpdftbl_error_handler_t hpdftbl_set_errhandler (
           hpdftbl_error_handler_t *err_handler*)

Specify errhandler for the table routines.

Note: The library provides a basic default error handler that can be used,

Parameters

err_handler	

Returns

    The old error handler or NULL if non exists

See also

    hpdftbl_default_table_error_handler()

16.12.3 Variable Documentation

16.12.3.1 hpdftbl_err_code

int hpdftbl_err_code = 0

Stores the last generated error code.

Internal variable to record last error

Referenced by hpdftbl_get_errstr(), and hpdftbl_get_last_errcode().

16.12.3.2 hpdftbl_err_col

int hpdftbl_err_col = -1

The column where the last error was generated.

Internal variable to record last error

16.12.3.3 hpdftbl_err_extrainfo

```
char hpdftbl_err_extrainfo[1024] = {0}
```

Extra info that may be specified at the point of error.

Internal variable to record last error

Referenced by hpdftbl_get_last_err_file().

16.12.3.4 hpdftbl_err_file

```
char* hpdftbl_err_file = NULL
```

Hold the file name where the last error occurred.

Internal variable to record last error

Referenced by hpdftbl_get_last_err_file().

16.12.3.5 hpdftbl_err_lineno

```
int hpdftbl_err_lineno = 0
```

Hold the line number of the last error occurred.

Internal variable to record last error

Referenced by hpdftbl_get_last_err_file().

16.12.3.6 hpdftbl_err_row

```
int hpdftbl_err_row = -1
```

The row where the last error was generated.

Internal variable to record last error

16.13 hpdftbl_grid.c File Reference

Create a grid on a document for positioning.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hpdf.h>
```

Functions

- void hpdftbl_stroke_grid (HPDF_Doc pdf, HPDF_Page page)

16.13.1 Detailed Description

Create a grid on a document for positioning.

16.13.2 Function Documentation

16.13.2.1 hpdftbl_stroke_grid()

```
void hpdftbl_stroke_grid (
            HPDF_Doc pdf,
            HPDF_Page page )
```

Stroke a point grid on specified page to make it easier to position text and tables.

Parameters

pdf	Document handle
page	Page handle

16.14 hpdftbl_load.c File Reference

Functions for load (internalizing) serialized data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <hpdf.h>
#include "hpdftbl.h"
#include <jansson.h>
```

Macros

- #define GETJSON_STRING(table, k, var)
- #define GETJSON_UINT(table, k, var)
- #define GETJSON_REAL(table, k, var)
- #define GETJSON_BOOLEAN(table, k, var)
- #define GETJSON_RGB(table, k, var)
- #define GETJSON_GRIDSTYLE(table, k, var)

- #define GETJSON_TXTSTYLE(table, k, var)
- #define GETJSON_REALARRAY(table, k, var)
- #define GETJSON_DYNCB(table, key)
- #define GETJSON_CELLDYNCB(table, key, r, c)
- #define GETJSON_CELLTXTSTYLE(table, key, r, c)

Functions

- int hpdftbl_theme_loads (hpdftbl_theme_t ∗theme, char ∗buff)

    Load theme from a serialized string. This is the invert function of hpdftbl_theme_dumps().
- int hpdftbl_theme_load (hpdftbl_theme_t ∗theme, char ∗filename)

    Read a theme from a previous serialized theme from a named file.
- int hpdftbl_load (hpdftbl_t tbl, char ∗filename)

    Import a table structure from a serialized table on file.
- int hpdftbl_loads (hpdftbl_t tbl, char ∗buff)

    Import a table structure from a serialized json buffert.

16.14.1 Detailed Description

Functions for load (internalizing) serialized data structure.

Author

Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

LICENSE

Released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN-CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.14.2 Macro Definition Documentation

16.14.2.1 GETJSON_BOOLEAN

```
#define GETJSON_BOOLEAN(
               table,
               k,
               var )
```

Value:
```
    do { \
    json_t *_elem=json_object_get(table,k); \
    if(!_elem) {                            \
        json_not_found_str=k;               \
        goto json_raise_notfound_error;     \
    }                                       \
    var=json_boolean_value(_elem);          \
} while(0)
```

16.14.2.2 GETJSON_CELLDYNCB

```
#define GETJSON_CELLDYNCB(
               table,
               key,
               r,
               c )
```

Value:
```
    do { \
    json_t *_elem=json_object_get(table,#key); \
    if(!_elem) { \
        json_not_found_str=#key; \
        goto json_raise_notfound_error; \
    } if( strlen(json_string_value(_elem)) != 0 ) \
        hpdftbl_set_cell_ ## key(t, r, c, json_string_value(_elem)); \
} while(0);
```

16.14.2.3 GETJSON_CELLTXTSTYLE

```
#define GETJSON_CELLTXTSTYLE(
               table,
               key,
               r,
               c )
```

Value:
```
    do { \
    json_t *__elem=json_object_get(table, #key); \
    if(!__elem) { \
        json_not_found_str= #key; \
        goto json_raise_notfound_error; \
    } \
    if( json_is_object(__elem) ) {  \
        GETJSON_STRING(__elem,"font",t->cells[t->cols*r+c].key.font); \
        GETJSON_REAL(__elem,"fsize",t->cells[t->cols*r+c].key.fsize); \
        GETJSON_RGB(__elem,"color",t->cells[t->cols*r+c].key.color);  \
        GETJSON_RGB(__elem,"background",t->cells[t->cols*r+c].key.background); \
        GETJSON_UINT(__elem,"halign",t->cells[t->cols*r+c].key.halign); \
    } else {                                        \
        json_not_found_str= #key;                   \
        goto json_raise_notfound_error;             \
    } \
} while(0);
```

16.14.2.4 GETJSON_DYNCB

```
#define GETJSON_DYNCB(
                table,
                key )
```

Value:
```
    do { \
    json_t *_elem=json_object_get(table, #key); \
    if(!_elem) { \
        json_not_found_str= #key; \
        goto json_raise_notfound_error; \
    } if( strlen(json_string_value(_elem)) != 0 ) \
        hpdftbl_set_ ## key(t, json_string_value(_elem)); \
} while(0);
```

16.14.2.5 GETJSON_GRIDSTYLE

```
#define GETJSON_GRIDSTYLE(
                table,
                k,
                var )
```

Value:
```
    do { \
    json_t *_grid=json_object_get(table,k); \
    if(!_grid) {                            \
        json_not_found_str=k;               \
        goto json_raise_notfound_error;     \
    } \
    if(json_is_object(_grid)) { \
        GETJSON_REAL(_grid,"width",var.width); \
        GETJSON_UINT(_grid,"dashstyle",var.line_dashstyle); \
        GETJSON_RGB(_grid,"color",var.color); \
    } \
} while(0)
```

16.14.2.6 GETJSON_REAL

```
#define GETJSON_REAL(
                table,
                k,
                var )
```

Value:
```
    do { \
    json_t *_elem=json_object_get(table,k); \
    if(!_elem) {                        \
        json_not_found_str=k;           \
        goto json_raise_notfound_error; \
    }                                   \
    var=json_real_value(_elem);         \
} while(0)
```

16.14.2.7 GETJSON_REALARRAY

```
#define GETJSON_REALARRAY(
                table,
                k,
                var )
```

Value:
```
    do { \
    json_t *_array = json_object_get(table, k); \
    if(!_array){ \
        json_not_found_str=k; \
        goto json_raise_notfound_error; \
    } \
    size_t _idx; \
    json_t *_val; \
    json_array_foreach(_array, _idx, _val) { \
        var[_idx] = json_real_value(_val); \
    } \
} while(0)
```

16.14.2.8 GETJSON_RGB

```
#define GETJSON_RGB(
                table,
                k,
                var )
```

Value:
```
    do { \
    json_t *_elem; \
    json_t *_val; \
    size_t _idx; \
    double tmpcol[3]; \
    _elem = json_object_get(table, k);\
    if(!_elem) {                        \
        json_not_found_str=k;           \
        goto json_raise_notfound_error; \
    } \
    json_array_foreach(_elem, _idx, _val) { \
        tmpcol[_idx] = json_real_value(_val); \
    } \
    var.r = tmpcol[0]; \
    var.g = tmpcol[1]; \
    var.b = tmpcol[2]; \
} while(0)
```

16.14.2.9 GETJSON_STRING

```
#define GETJSON_STRING(
                table,
                k,
                var )
```

Value:
```
    do  { \
    json_t *_elem=json_object_get(table,k); \
    if(!_elem) {                            \
        json_not_found_str=k;               \
        goto json_raise_notfound_error;     \
    }                                       \
    if( strlen(json_string_value(_elem)) == 0 ) \
        var=NULL;                           \
    else                                    \
        var=strdup(json_string_value(_elem)); \
} while(0)
```

16.14.2.10 GETJSON_TXTSTYLE

```
#define GETJSON_TXTSTYLE(
            table,
            k,
            var )
```

Value:
```
    do { \
    json_t *_txtstyle=json_object_get(table,k); \
    if(!_txtstyle) {                            \
        json_not_found_str=k;                   \
        goto json_raise_notfound_error;         \
    } \
    if(json_is_object(_txtstyle)) { \
        GETJSON_STRING(_txtstyle,"font",var.font); \
        GETJSON_REAL(_txtstyle,"fsize",var.fsize); \
        GETJSON_RGB(_txtstyle,"color",var.color); \
        GETJSON_RGB(_txtstyle,"background",var.background); \
        GETJSON_UINT(_txtstyle,"halign",var.halign); \
    } \
} while(0)
```

16.14.2.11 GETJSON_UINT

```
#define GETJSON_UINT(
            table,
            k,
            var )
```

Value:
```
    do { \
    json_t *_elem=json_object_get(table,k); \
    if(!_elem) {                            \
        json_not_found_str=k;               \
        goto json_raise_notfound_error;     \
    }                                       \
    var=(size_t)json_integer_value(_elem);  \
} while(0)
```

16.14.3 Function Documentation

16.14.3.1 hpdftbl_load()

```
int hpdftbl_load (
            hpdftbl_t tbl,
            char * filename )
```

Import a table structure from a serialized table on file.

The json file make it possible to adjust the table directly. However it is easy to get it wrong. Some things to keep in mind while doing manual changes.

- A real number must always be written as a decimal number with at least one decimal point (even if it .0)

- Remember that the width of the table is specified manually and not automatically recalculated based on the text width.

After reading a serialized table it can asily be be stroked with only two lines of code as the following code-snippet shows

```
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
if(0 == hpdftbl_load(tbl, "mytablefile.json") ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

Note

The hpdftbl_t is a pointer type to struct hpdftbl and hence must must be either dynamically allocated as the example here shows or an instance of the struct must be created whose address is given to this functions.

Parameters

tbl	Table to read into
filename	File to read from

Returns

0 on success, -1 on file parse error, -2 on nay other error

16.14.3.2 hpdftbl_loads()

```
int hpdftbl_loads (
            hpdftbl_t tbl,
            char * buff )
```

Import a table structure from a serialized json buffert.

This is the preferred way on how to store a table structure in for example a database.

Example:
```
char *mybuffer = ....
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
if(0 == hpdftbl_load(tbl, mybuffer) ) {
    hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
}
```

Parameters

tbl	Reference to table handle to be populated
buff	Buffer with serialized data to read back

Returns

0 on success, -1 on file parse error, -2 on nay other error

See also

[hpdftbl_dump()](), [hpdftbl_load()](), [hpdftbl_stroke_pos()]()

Referenced by [hpdftbl_load()]().

16.14.3.3 hpdftbl_theme_load()

```
int hpdftbl_theme_load (
            hpdftbl_theme_t * theme,
            char * filename )
```

Read a theme from a previous serialized theme from a named file.

Example:
```
hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
hpdftbl_theme_t theme;
if( 0 == hpdftbl_load(tbl, "tests/tut_ex41.json") ) {
    if( 0 == hpdftbl_theme_load(&theme, "mytheme.json") ) {
        hpdftbl_apply_theme(tbl, &theme);
        hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
    }
}
```

Parameters

theme	Theme to read into
filename	File to read from

Returns

0 on success, -1 on failure

16.14.3.4 hpdftbl_theme_loads()

```
int hpdftbl_theme_loads (
            hpdftbl_theme_t * theme,
            char * buff )
```

Load theme from a serialized string. This is the invert function of [hpdftbl_theme_dumps()]().

Parameters

theme	Theme to load to.
buff	Buffer which holds the previous serialized theme

Returns

0 on success, -1 on failure

See also

> hpdftbl_theme_dumps(), hpdftbl_theme_load(), hpdftbl_apply_theme()

Referenced by hpdftbl_theme_load().

16.15 hpdftbl_theme.c File Reference

Functions for theme handling.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <hpdf.h>
#include "hpdftbl.h"
```

Macros

- #define **HPDFTBL_DEFAULT_TITLE_STYLE** (hpdf_text_style_t){HPDF_FF_HELVETICA_BOLD,11,(HPDF↵
  _RGBColor){0,0,0},(HPDF_RGBColor){0.9f,0.9f,0.9f}, LEFT}

  Default style for table title.
- #define HPDFTBL_DEFAULT_HEADER_STYLE (hpdf_text_style_t){HPDF_FF_HELVETICA_BOLD,10,(HPDF↵
  _RGBColor){0,0,0},(HPDF_RGBColor){0.9f,0.9f,0.97f}, CENTER}

  Default style for table header row.
- #define HPDFTBL_DEFAULT_LABEL_STYLE (hpdf_text_style_t){HPDF_FF_TIMES_ITALIC,9,(HPDF_↵
  RGBColor){0.4f,0.4f,0.4f},(HPDF_RGBColor){1,1,1}, LEFT}

  Default style for table header row.
- #define HPDFTBL_DEFAULT_CONTENT_STYLE (hpdf_text_style_t){HPDF_FF_COURIER,10,(HPDF_↵
  RGBColor){0.2f,0.2f,0.2f},(HPDF_RGBColor){1,1,1}, LEFT}

  Default style for table header row.
- #define HPDFTBL_DEFAULT_INNER_VGRID_STYLE (hpdftbl_grid_style_t){0.7, (HPDF_RGBColor){0.↵
  5f,0.5f,0.5f},0}

  Default style for table vertical inner grid.
- #define HPDFTBL_DEFAULT_INNER_HGRID_STYLE (hpdftbl_grid_style_t){0.7, (HPDF_RGBColor){0.↵
  5f,0.5f,0.5f},0}

  Default style for table horizontal inner grid.
- #define HPDFTBL_DEFAULT_OUTER_GRID_STYLE (hpdftbl_grid_style_t){1.0f, (HPDF_RGBColor){0.↵
  2f,0.2f,0.2f},0}

  Default style for table outer grid (border)
- #define **HPDFTBL_DEFAULT_ZEBRA_COLOR1** HPDF_COLOR_WHITE

  Default style for alternating row backgrounds color 1.
- #define **HPDFTBL_DEFAULT_ZEBRA_COLOR2** HPDF_COLOR_XLIGHT_GRAY

  Default style for alternating row backgrounds color 2.

Functions

- int hpdftbl_apply_theme (hpdftbl_t t, hpdftbl_theme_t ∗theme)

  Apply a specified theme to a table.
- int hpdftbl_get_theme (hpdftbl_t tbl, hpdftbl_theme_t ∗theme)

  Extract theme from settings of a specific table.
- hpdftbl_theme_t ∗ hpdftbl_get_default_theme (void)

  Return the default theme.
- int hpdftbl_destroy_theme (hpdftbl_theme_t ∗theme)

  Destroy existing theme structure and free memory.

16.15.1 Detailed Description

Functions for theme handling.

Author

> Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

> LICENSE

Released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN-CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.15.2 Macro Definition Documentation

16.15.2.1 HPDFTBL_DEFAULT_CONTENT_STYLE

```
#define HPDFTBL_DEFAULT_CONTENT_STYLE (hpdf_text_style_t){HPDF_FF_COURIER,10,(HPDF_RGBColor){0.↩
2f,0.2f,0.2f},(HPDF_RGBColor){1,1,1}, LEFT}
```

Default style for table header row.

See also

> hpdftbl_set_content_style()

16.15.2.2 HPDFTBL_DEFAULT_HEADER_STYLE

`#define HPDFTBL_DEFAULT_HEADER_STYLE (hpdf_text_style_t){HPDF_FF_HELVETICA_BOLD,10,(HPDF_↩`
`RGBColor){0,0,0},(HPDF_RGBColor){0.9f,0.9f,0.97f}, CENTER}`

Default style for table header row.

See also

> hpdftbl_set_header_style()

16.15.2.3 HPDFTBL_DEFAULT_INNER_HGRID_STYLE

`#define HPDFTBL_DEFAULT_INNER_HGRID_STYLE (hpdftbl_grid_style_t){0.7, (HPDF_RGBColor){0.↩`
`5f,0.5f,0.5f},0}`

Default style for table horizontal inner grid.

See also

> hpdftbl_set_inner_hgrid_style()

16.15.2.4 HPDFTBL_DEFAULT_INNER_VGRID_STYLE

`#define HPDFTBL_DEFAULT_INNER_VGRID_STYLE (hpdftbl_grid_style_t){0.7, (HPDF_RGBColor){0.↩`
`5f,0.5f,0.5f},0}`

Default style for table vertical inner grid.

See also

> hpdftbl_set_inner_vgrid_style()

16.15.2.5 HPDFTBL_DEFAULT_LABEL_STYLE

`#define HPDFTBL_DEFAULT_LABEL_STYLE (hpdf_text_style_t){HPDF_FF_TIMES_ITALIC,9,(HPDF_RGBColor){0.↩`
`4f,0.4f,0.4f},(HPDF_RGBColor){1,1,1}, LEFT}`

Default style for table header row.

See also

> hpdftbl_set_label_style()

16.15.2.6 HPDFTBL_DEFAULT_OUTER_GRID_STYLE

```
#define HPDFTBL_DEFAULT_OUTER_GRID_STYLE (hpdftbl_grid_style_t){1.0f, (HPDF_RGBColor){0.↩
2f,0.2f,0.2f},0}
```

Default style for table outer grid (border)

See also

> hpdftbl_set_outer_grid_style()

16.15.3 Function Documentation

16.15.3.1 hpdftbl_apply_theme()

```
int hpdftbl_apply_theme (
             hpdftbl_t t,
             hpdftbl_theme_t * theme )
```

Apply a specified theme to a table.

Note however that a limitation (by design) of themes is that settings in individual cells are not recorded in a theme since a theme can be applied to any table despite the structure. This mean only settings that are generic to a table is stored in a theme. Not individal cells.

The default table theme can be retrieved with hpdftbl_get_default_theme()

Parameters

t	Table handle
theme	Theme reference

Returns

> 0 on success, -1 on failure

See also

> hpdftbl_get_default_theme()

Referenced by hpdftbl_create_title(), and hpdftbl_stroke_from_data().

16.15.3.2 hpdftbl_destroy_theme()

```
int hpdftbl_destroy_theme (
             hpdftbl_theme_t * theme )
```

Destroy existing theme structure and free memory.

Free all memory allocated by a theme

Parameters

theme	The theme to free

Returns

> -1 for error , 0 for success

Referenced by hpdftbl_create_title().

16.15.3.3 hpdftbl_get_default_theme()

```
hpdftbl_theme_t * hpdftbl_get_default_theme (
            void  )
```

Return the default theme.

Create and return a theme corresponding to the default table theme. It is the calling functions responsibility to call hpdftbl_destroy_theme() to free the allocated memory. The default theme is a good starting point to just make minor modifications without having to define all elements.

Returns

> A new theme initialized to the default settings. It is the calling routines responsibility to free memory used in the returned theme with hpdftbl_destroy_theme()

See also

> hpdftbl_apply_theme(), hpdftbl_destroy_theme()

Referenced by hpdftbl_create_title().

16.15.3.4 hpdftbl_get_theme()

```
int hpdftbl_get_theme (
            hpdftbl_t tbl,
            hpdftbl_theme_t * theme )
```

Extract theme from settings of a specific table.

This is useful if a table has been specified with some specific look & feel and another table should be given the same l&f.

Note however that a limitation (by design) of themes is that settings in individual cells are not recorded in a theme since a theme can be applied to any table despite the structure. This mean only settings that are generic to a table is stored in a theme. Not individal cells.

Parameters

tbl	Table handle for table to have its settings extracted
theme	Theme to be read out to.

Returns

0 on success, -1 on failure

16.16 hpdftbl_widget.c File Reference

Support for drawing widgets.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <hpdf.h>
#include <string.h>
#include <math.h>
#include "hpdftbl.h"
```

Macros

- #define TRUE 1
- #define FALSE 0

Functions

- void hpdftbl_table_widget_letter_buttons (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF←
  _REAL ypos, HPDF_REAL width, HPDF_REAL height, const HPDF_RGBColor on_color, const HPDF_←
  RGBColor off_color, const HPDF_RGBColor on_background, const HPDF_RGBColor off_background, const
  HPDF_REAL fsize, const char *letters, _Bool *state)

  Display an array of letters as a table where each letter is its own "mini" cell
  and sorrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different
  font and fac colors.

- void hpdftbl_widget_slide_button (HPDF_Doc doc, HPDF_Page page, HPDF_REAL xpos, HPDF_REAL
  ypos, HPDF_REAL width, HPDF_REAL height, _Bool state)

  Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.

- void hpdftbl_widget_hbar (const HPDF_Doc doc, const HPDF_Page page, const HPDF_REAL xpos, const
  HPDF_REAL ypos, const HPDF_REAL width, const HPDF_REAL height, const HPDF_RGBColor color,
  const float val, const _Bool hide_val)

  Draw a horizontal partially filled bar to indicate an analog (percentage) value.

- void hpdftbl_widget_segment_hbar (const HPDF_Doc doc, const HPDF_Page page, const HPDF_REAL
  xpos, const HPDF_REAL ypos, const HPDF_REAL width, const HPDF_REAL height, const size_t num←
  _segments, const HPDF_RGBColor on_color, const double val_percent, const _Bool hide_val)

  Draw a horizontal segment meter that can be used to visualize a discrete value.

- void hpdftbl_widget_strength_meter (const HPDF_Doc doc, const HPDF_Page page, const HPDF_REAL
  xpos, const HPDF_REAL ypos, const HPDF_REAL width, const HPDF_REAL height, const size_t num_←
  segments, const HPDF_RGBColor on_color, const size_t num_on_segments)

  Draw a phone strength meter.

16.16.1 Detailed Description

Support for drawing widgets.

16.16.2 Macro Definition Documentation

16.16.2.1 FALSE

```
#define FALSE 0
```

C Boolean false value

16.16.2.2 TRUE

```
#define TRUE 1
```

C Boolean truth value

16.16.3 Function Documentation

16.16.3.1 hpdftbl_table_widget_letter_buttons()

```
void hpdftbl_table_widget_letter_buttons (
            HPDF_Doc doc,
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            HPDF_REAL width,
            HPDF_REAL height,
            const HPDF_RGBColor on_color,
            const HPDF_RGBColor off_color,
            const HPDF_RGBColor on_background,
            const HPDF_RGBColor off_background,
            const HPDF_REAL fsize,
            const char * letters,
            _Bool * state )
```

Display an array of letters as a table where each letter is its own "mini" cell
and sorrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different font and fac colors.

Parameters

doc	HPDF document handle
page	HPDF page handle

Parameters

xpos	X-öosition of cell
ypos	Y-Position of cell
width	Width of cell
height	Height of cell
on_color	The font color in "on" state
off_color	The font color in "off" state
on_background	The face color in "on" state
off_background	The face color in "off" state
fsize	The font size
letters	What letters to have in the boxes
state	What state each boxed letter should be (0=off, 1=pn)

16.16.3.2 hpdftbl_widget_hbar()

```
void hpdftbl_widget_hbar (
            const HPDF_Doc doc,
            const HPDF_Page page,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            const HPDF_REAL height,
            const HPDF_RGBColor color,
            const float val,
            const _Bool hide_val )
```

Draw a horizontal partially filled bar to indicate an analog (percentage) value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

Parameters

doc	HPDF Document handle
page	HPDF Page handle
xpos	Lower left x
ypos	Lower left y
width	Width of meter
height	Height of meter
color	Fill color for bar
val	Percentage fill in range [0.0, 100.0]
hide_val	TRUE to hide the value (in percent) at the right end of the entire bar

16.16.3.3 hpdftbl_widget_segment_hbar()

```
void hpdftbl_widget_segment_hbar (
            const HPDF_Doc doc,
            const HPDF_Page page,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            const HPDF_REAL height,
            const size_t num_segments,
            const HPDF_RGBColor on_color,
            const double val_percent,
            const _Bool hide_val )
```

Draw a horizontal segment meter that can be used to visualize a discrete value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

Parameters

doc	HPDF Document handle
page	HPDF Page handle
xpos	Lower left x
ypos	Lower left y
width	Width of meter
height	Height of meter
num_segments	Total number of segments
on_color	Color for "on" segment
val_percent	To what extent should the bars be filled (as a value 0.0 - 1.0)
hide_val	TRUE to hide the value (in percent) at the right end of the entire bar

16.16.3.4 hpdftbl_widget_slide_button()

```
void hpdftbl_widget_slide_button (
            HPDF_Doc doc,
            HPDF_Page page,
            HPDF_REAL xpos,
            HPDF_REAL ypos,
            HPDF_REAL width,
            HPDF_REAL height,
            _Bool state )
```

Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.

This function can not be used directly as a canvas callback since it needs the state of the button as an argument. Instead create a simple canvas callback that determines the wanted state and then just passes on all argument to this widget function.

Parameters

doc	HPDF document handle
page	HPDF page handle
xpos	X-öosition of cell
ypos	Y-Position of cell
width	Width of cell
height	Height of cell
state	State of button On/Off

16.16.3.5 hpdftbl_widget_strength_meter()

```
void hpdftbl_widget_strength_meter (
            const HPDF_Doc doc,
            const HPDF_Page page,
            const HPDF_REAL xpos,
            const HPDF_REAL ypos,
            const HPDF_REAL width,
            const HPDF_REAL height,
            const size_t num_segments,
            const HPDF_RGBColor on_color,
            const size_t num_on_segments )
```

Draw a phone strength meter.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

Parameters

doc	HPDF Document handle
page	HPDF Page handle
xpos	Lower left x
ypos	Lower left y
width	Width of meter
height	Height of meter
num_segments	Total number of segments
on_color	Color for "on" segment
num_on_segments	Number of on segments

16.17 read_file.c File Reference

Function for reading a file into a memory buffer.

```
#include <stdio.h>
#include <hpdf.h>
#include "hpdftbl.h"
```

Functions

- int hpdftbl_read_file (char ∗buff, size_t buffsize, char ∗filename)

  Read content of file into a specified buffer.

16.17.1 Detailed Description

Function for reading a file into a memory buffer.

Author

Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

LICENSE

Released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN-CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.17.2 Function Documentation

16.17.2.1 hpdftbl_read_file()

```
int hpdftbl_read_file (
            char * buff,
            size_t buffsize,
            char * filename )
```

Read content of file into a specified buffer.

Parameters

buff	Destination buffer
buffsize	Size of buffer
filename	Name of file to read from

Returns

-1 on failure, 0 on success

Referenced by hpdftbl_load(), and hpdftbl_theme_load().

16.18   xstr.c File Reference

Safe version of strncat() and strncpy() taken from the BSD stdlib.

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
```

Functions

- size_t xstrlcpy (char ∗__restrict dst, const char ∗__restrict src, size_t dsize)

    Safe string copy.
- size_t xstrlcat (char ∗dst, const char ∗src, size_t siz)

    Safe string concatenation.

16.18.1   Detailed Description

Safe version of strncat() and strncpy() taken from the BSD stdlib.

16.18.2   Function Documentation

16.18.2.1   xstrlcat()

```
size_t xstrlcat (
            char * dst,
            const char * src,
            size_t siz )
```

Safe string concatenation.

Appends src to string dst of size siz (unlike strncat, siz is the full size of dst, not space left). At most siz-1 characters will be copied. Always NUL terminates (unless siz <= strlen(dst)). Returns strlen(src) + MIN(siz, strlen(initial dst)). If retval >= siz, truncation occurred.

Taken from BSD library.

Parameters

dst	Destination buffer
src	Source buffer
siz	Max size of destination buffer including terminating NULL

Returns

The number of bytes needed to be copied. If this is $>$ `siz` then data truncation happened.

Referenced by hpdftbl_read_file().

16.18.2.2 xstrlcpy()

```
size_t xstrlcpy (
            char *__restrict dst,
            const char *__restrict src,
            size_t dsize )
```

Safe string copy.

Copy string src to buffer dst of size dsize. At most dsize-1 chars will be copied. Always NUL terminates (unless dsize == 0). Returns strlen(src); if retval >= dsize, truncation occurred.

Taken from BSD library.

Parameters

dst	Destination string
src	Source string
dsize	Maximum size of destination

Returns

strlen(src); if retval >= dsize, truncation occurred.

Chapter 17

Example Documentation

17.1 example01.c

A collection of more and less advanced examples in one file. For learning the library it is better to start with the organized tutorial examples like tut_ex01.c and tut_ex02.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#include <sys/stat.h>
#include <libgen.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// The output after running the program will be written to this file
#ifdef _WIN32
#define OUTPUT_FILE "example01.pdf"
#else
#define OUTPUT_FILE "/tmp/example01.pdf"
#endif
// For simulated exception handling
jmp_buf _hpdftbl_jmp_env;
#include "unit_test.inc.h"
// Global handlers to the HPDF document and page
HPDF_Doc pdf_doc;
HPDF_Page pdf_page;
// We use some dummy data to populate the tables
#define MAX_NUM_ROWS 10
#define MAX_NUM_COLS 10
// Data array with string pointers to dummy data and cell labels
// The actual storage for the strings are dynamically allocated.
char *labels[MAX_NUM_ROWS * MAX_NUM_COLS];
char *content[MAX_NUM_ROWS * MAX_NUM_COLS];
// Create two arrays with dummy data to populate the tables
void
setup_dummy_data(void) {
    char buff[255];
    size_t cnt = 0;
    for (size_t r = 0; r < MAX_NUM_ROWS; r++) {
        for (size_t c = 0; c < MAX_NUM_COLS; c++) {
#if (defined _WIN32 || defined __WIN32__)
            sprintf(buff, "Label %i:", cnt);
            labels[cnt] = _strdup(buff);
            sprintf(buff, "Content %i", cnt);
            content[cnt] = _strdup(buff);
#else
            snprintf(buff, sizeof(buff), "Label %zu:", cnt);
            labels[cnt] = strdup(buff);
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            content[cnt] = strdup(buff);
```

```
#endif
            cnt++;
        }
    }
}
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
#if !(defined _WIN32 || defined __WIN32__)
// We don't use the page header on Windooze systems
static char *
cb_name(void *tag, size_t r, size_t c) {
    static char buf[256];
    struct utsname sysinfo;
    if (run_as_unit_test || -1 == uname(&sysinfo)) {
        return "???";
    } else {
        snprintf(buf, sizeof(buf), "Name: %s, Kernel: %s %s", sysinfo.nodename,
                 sysinfo.sysname, sysinfo.release);
        return buf;
    }
}
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
void
cb_draw_segment_hbar(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                     size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                     HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL segment_tot_width = width * 0.5;
    const HPDF_REAL segment_height = height / 3;
    const HPDF_REAL segment_xpos = xpos + 40;
    const HPDF_REAL segment_ypos = ypos + 4;
    const size_t num_segments = 10;
    const HPDF_RGBColor on_color = HPDF_COLOR_GREEN;
    const double val_percent = 0.4;
    const _Bool val_text_hide = FALSE;
    hpdftbl_widget_segment_hbar(
            doc, page, segment_xpos, segment_ypos, segment_tot_width,
            segment_height, num_segments, on_color, val_percent, val_text_hide);
}
void
cb_draw_hbar(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r, size_t c,
             HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width,
             HPDF_REAL height) {
    const HPDF_REAL wwidth = width * 0.5;
    const HPDF_REAL wheight = height / 3;
    const HPDF_REAL wxpos = xpos + 40;
    const HPDF_REAL wypos = ypos + 4;
    const HPDF_RGBColor color = HPDF_COLOR_GREEN;
    const double val = 0.6;
    const _Bool val_text_hide = FALSE;
    hpdftbl_widget_hbar(doc, page, wxpos, wypos, wwidth, wheight, color, val,
                        val_text_hide);
}
void
cb_draw_slider(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r, size_t c,
               HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width,
               HPDF_REAL height) {
    /*
     * void
hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL
height, _Bool state)
     */
    const HPDF_REAL wwidth = 37;
    const HPDF_REAL wheight = 12;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 5;
    // The slide is on for third row and off otherwise
    _Bool state = (r == 2);
    hpdftbl_widget_slide_button(doc, page, wxpos, wypos, wwidth, wheight,
                                state);
}
void
cb_draw_strength_meter(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                       size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
```

```
                                HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL wwidth = 35;
    const HPDF_REAL wheight = 20;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 4;
    const size_t num_segments = 5;
    const HPDF_RGBColor on_color = HPDF_COLOR_GREEN;
    const size_t num_on_segments = 3;
    hpdftbl_widget_strength_meter(doc, page, wxpos, wypos, wwidth, wheight,
                                  num_segments, on_color, num_on_segments);
}
void
cb_draw_boxed_letter(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                     size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                     HPDF_REAL width, HPDF_REAL height) {
    /*
     * void
     hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
                        HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL
                        height, const HPDF_RGBColor on_color, const HPDF_RGBColor off_color, const
                        HPDF_RGBColor on_background, const HPDF_RGBColor off_background, const HPDF_REAL
                        fsize, const char *letters, _Bool *state )
     */
    const HPDF_REAL wwidth = 60;
    const HPDF_REAL wheight = 15;
    const HPDF_REAL wxpos = xpos + 60;
    const HPDF_REAL wypos = ypos + 4;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_GRAY;
    const HPDF_RGBColor off_color = HPDF_COLOR_GRAY;
    const HPDF_RGBColor on_background = HPDF_COLOR_GREEN;
    const HPDF_RGBColor off_background = HPDF_COLOR_LIGHT_GRAY;
    const HPDF_REAL fsize = 11;
    const char *letters = "ABCD";
    _Bool state[] = {TRUE, FALSE, TRUE, FALSE};
    hpdftbl_table_widget_letter_buttons(doc, page, wxpos, wypos, wwidth, wheight,
                                  on_color, off_color, on_background,
                                  off_background, fsize, letters, state);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
example_page_header(void) {
    // Specified the layout of each row
    // For a cell where we want dynamic content we must make use of a
    // content-callback that will return a pointer to a static buffer whose
    // content will be displayed in the cell.
    hpdftbl_cell_spec_t tbl1_data[] = {
            // row,col,rowspan,colspan,lable-string,content-callback
            {0, 0, 1, 4, "Server info:", cb_name, NULL, NULL, NULL},
            {0, 4, 1, 2, "Date:",        cb_date, NULL, NULL, NULL},
            {0, 0, 0, 0, NULL, NULL,              NULL, NULL, NULL} /* Sentinel to mark end of data */
    };
    // Overall table layout
    hpdftbl_spec_t tbl1 = {
            .title=NULL, .use_header=0,
            .use_labels=1, .use_labelgrid=1,
            .rows=1, .cols=6,
            .xpos=50, .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1),
            .width=500, .height=0,
            .content_cb=0, .label_cb=0, .style_cb=0, .post_cb=0,
            .cell_spec=tbl1_data
    };
    // Show how to set a specified theme to the table. Since we only use the
    // default theme here we could equally well just have set NULL as the last
    // argument to the hpdftbl_stroke_from_data() function since this is the
    // same specifying the default theme.
    hpdftbl_theme_t *theme = hpdftbl_get_default_theme();
    int ret = hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl1, theme);
    // Should always check for any error
    if (-1 == ret) {
        const char *buf;
        int r, c;
        int tbl_err = hpdftbl_get_last_errcode(&buf, &r, &c);
        fprintf(stderr,
                "*** ERROR in creating table from data. ( %d : \"%s\" ) @ "
                "[%d,%d]\n",
                tbl_err, buf, r, c);
    }
    // Remember to clean up to avoid memory leak
    hpdftbl_destroy_theme(theme);
}
#endif
// Add another page in the document
static void
add_a4page(void) {
    pdf_page = HPDF_AddPage(pdf_doc);
```

```c
    HPDF_Page_SetSize(pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
}
void
ex_tbl1(void) {
    int num_rows = 5;
    int num_cols = 4;
    char *table_title = "Example 1: Basic table with default theme";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, FALSE);
    //hpdftbl_use_labelgrid(t, TRUE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_tbl2(void) {
    int num_rows = 5;
    int num_cols = 4;
    char *table_title = "Example 2: Basic table with adjusted font styles";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                            title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    // Use bold font for content. Use the C99 way to specify constant structure
    // constants
    const HPDF_RGBColor content_text_color = HPDF_COLOR_DARK_GRAY;
    const HPDF_RGBColor content_bg_color = HPDF_COLOR_LIGHT_BLUE;
    hpdftbl_set_content_style(t, HPDF_FF_COURIER_BOLD, 10,
                              content_text_color, content_bg_color);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, TRUE);
    hpdftbl_use_labelgrid(t, TRUE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_tbl3(void) {
    int num_rows = 9;
    int num_cols = 4;
    char *table_title =
            "Example 3: Table cell spannings and full grid and header";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                            title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    // Use specially formatted header row
    hpdftbl_use_header(t, TRUE);
    // Use full grid and not just the short labelgrid
    hpdftbl_use_labelgrid(t, FALSE);
    // Use bold font for content. Use the C99 way to specify constant structure
    // constants
    const HPDF_RGBColor content_text_color = HPDF_COLOR_DARK_GRAY;
    const HPDF_RGBColor content_bg_color = HPDF_COLOR_WHITE;
    hpdftbl_set_content_style(t, HPDF_FF_COURIER_BOLD, 10,
                              content_text_color, content_bg_color);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, TRUE);
    // Spanning for the header row (row==0))
    // Span cell=(0,1) one row and three columns
    hpdftbl_set_cellspan(t, 0, 1, 1, 3);
    // Span cell=(1,1) one row and three columns
    hpdftbl_set_cellspan(t, 1, 1, 1, 3);
    // Span cell=(2,2) one row and two columns
    hpdftbl_set_cellspan(t, 2, 2, 1, 2);
    // Span cell=(4,1) two rows and three columns
    hpdftbl_set_cellspan(t, 4, 1, 2, 3);
```

```
    // Span cell=(7,2) two rows and two columns
    hpdftbl_set_cellspan(t, 7, 2, 2, 2);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_tbl4(void) {
    const size_t num_rows = 5;
    const size_t num_cols = 4;
    char *table_title = "Example 4: Adjusting look and feel of single cell";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                            title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    // Set the top left and bottom right with orange bg_color
    const HPDF_RGBColor content_bg_color = HPDF_COLOR_ORANGE;
    const HPDF_RGBColor content_text_color = HPDF_COLOR_ALMOST_BLACK;
    hpdftbl_set_cell_content_style(t, 0, 0, HPDF_FF_COURIER_BOLD, 10,
                                   content_text_color, content_bg_color);
    hpdftbl_set_cell_content_style(t, 4, 3, HPDF_FF_COURIER_BOLD, 10,
                                   content_text_color, content_bg_color);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, TRUE);
    hpdftbl_use_labelgrid(t, TRUE);
    // First column should be 40% of the total width
    hpdftbl_set_colwidth_percent(t, 0, 40);
    // Span cell=(1,0) one row and two columns
    hpdftbl_set_cellspan(t, 1, 0, 1, 2);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0;  // Calculate height automatically
    if (-1 ==
        hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height)) {
        const char *errstr;
        int row, col;
        hpdftbl_get_last_errcode(&errstr, &row, &col);
        fprintf(stderr, "ERROR: \"%s\"\n", errstr);
    }
}
void
ex_tbl5(void) {
    const int num_rows = 6;
    const int num_cols = 4;
    char *table_title = "Example 5: Using widgets in cells";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                            title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    hpdftbl_set_min_rowheight(t, 20);
    // Install callback for the specified cell where the graphical meter will be
    // drawn
    size_t wrow = 0;
    size_t wcol = 0;
    content[wrow * num_cols + wcol] = NULL;
    labels[wrow * num_cols + wcol] = "Horizontal seg bar:";
    hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_segment_hbar);
    wrow += 1;
    content[wrow * num_cols + wcol] = NULL;
    labels[wrow * num_cols + wcol] = "Horizontal bar:";
    hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_hbar);
    wrow += 1;
    content[wrow * num_cols + wcol] = NULL;
    labels[wrow * num_cols + wcol] = "Slider on:";
    hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_slider);
    wrow += 1;
    content[wrow * num_cols + wcol] = NULL;
    labels[wrow * num_cols + wcol] = "Slider off:";
    hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_slider);
    wrow += 1;
    content[wrow * num_cols + wcol] = NULL;
```

```c
        labels[wrow * num_cols + wcol] = "Strength meter:";
        hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_strength_meter);
        wrow += 1;
        content[wrow * num_cols + wcol] = NULL;
        labels[wrow * num_cols + wcol] = "Boxed letters:";
        hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_boxed_letter);
        hpdftbl_set_content(t, content);
        hpdftbl_set_labels(t, labels);
        hpdftbl_use_labels(t, TRUE);
        hpdftbl_use_labelgrid(t, TRUE);
        // First column should be 40% of the total width
        hpdftbl_set_colwidth_percent(t, 0, 40);
        // We let the library automatically determine the height of the table based
        // on the font and number of rows.
        HPDF_REAL xpos = hpdftbl_cm2dpi(2);
        HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
        HPDF_REAL width = hpdftbl_cm2dpi(15);
        HPDF_REAL height = 0;  // Calculate height automatically
        if (-1 ==
            hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height)) {
            const char *errstr;
            int row, col;
            hpdftbl_get_last_errcode(&errstr, &row, &col);
            fprintf(stderr, "ERROR: \"%s\"\n", errstr);
        }
}
// Type for the pointer to example stroking functions "void fnc(void)"
typedef void (*t_func_tbl_stroke)(void);
int
main(int argc, char **argv) {
    t_func_tbl_stroke examples[] = {ex_tbl1, ex_tbl2, ex_tbl3, ex_tbl4,
                                    ex_tbl5};
    const size_t num_examples = sizeof(examples) / sizeof(t_func_tbl_stroke);
    printf("Stroking %ld examples.\n", num_examples);
    // Setup fake exception handling
    if (setjmp(_hpdftbl_jmp_env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    run_as_unit_test = 2 == argc ;
    // Get some dummy data to fill the table$
    setup_dummy_data();
    // Setup the basic PDF document
    pdf_doc = HPDF_New(error_handler, NULL);
    HPDF_SetCompressionMode(pdf_doc, HPDF_COMP_ALL);
    for (size_t i = 0; i < num_examples; i++) {
        add_a4page();
#if !(defined _WIN32 || defined __WIN32__)
        example_page_header();
#endif
        (*examples[i])();
    }
    if( -1 == stroke_to_file(pdf_doc, argc, argv) )
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}
```

17.2 tut_ex00.c

The very most basic table with a header

```c
#include "unit_test.inc.h"
void
create_table_ex00(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    int num_rows = 4;
    int num_cols = 3;
    char *table_title = "Example 1: Basic table with default theme";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);
    hpdftbl_use_labels(tbl, TRUE);
    // hpdftbl_use_labelgrid(tbl, TRUE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
```

```
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 8);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex00, FALSE)
```

17.3 tut_ex01.c

The very most basic table with API call to set content in each cell.

```
#include "unit_test.inc.h"
void
create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
    hpdftbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
    hpdftbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
    hpdftbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex01, FALSE)
```

17.4 tut_ex02.c

Basic table with content data specified as an array.

```
#include "unit_test.inc.h"
void
create_table_ex02(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, 2, 2);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex02, FALSE)
```

17.5 tut_ex02_1.c

Basic table with content data specified as an array.

```
#include "unit_test.inc.h"
void setup_dummy_content_with_header(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            if( 0==r )
                snprintf(buff, sizeof(buff), "Header %zu", cnt);
            else
                snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
```

```
    }
}
void
create_table_ex02_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    content_t content;
    setup_dummy_content_with_header(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex02_1, FALSE)
```

17.6   tut_ex03.c

First example with API call to set content in each cell with added labels and shortened grid.

```
#include "unit_test.inc.h"
void
create_table_ex03(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_cell(tbl, 0, 0, "Label 1:", "Cell 0x0");
    hpdftbl_set_cell(tbl, 0, 1, "Label 2:", "Cell 0x1");
    hpdftbl_set_cell(tbl, 1, 0, "Label 3:", "Cell 1x0");
    hpdftbl_set_cell(tbl, 1, 1, "Label 4:", "Cell 1x1");
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, FALSE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex03, FALSE)
```

17.7   tut_ex04.c

Specifying labels as data array.

```
#include "unit_test.inc.h"
void
create_table_ex04(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);

    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex04, FALSE)
```

17.8 tut_ex05.c

Set content data specified as an array with added labels and shortened grid.

```
#include "unit_test.inc.h"
void
create_table_ex05(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex05: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex05, FALSE)
```

17.9 tut_ex06.c

Use content to set content and labels.

```
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    snprintf(buf, sizeof buf, "Content %02i x %02i", r, c);
#else
    snprintf(buf, sizeof buf, "Content %02zu x %02zu", r, c);
#endif
    return buf;
}
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
    }
#else
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
#endif
    return buf;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex06(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
```

```
        const size_t num_rows = 2;
        const size_t num_cols = 2;
        char *table_title = "tut_ex06: 2x2 table with callbacks";
        hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
        hpdftbl_use_labels(tbl, TRUE);
        hpdftbl_use_labelgrid(tbl, TRUE);
        hpdftbl_set_content_cb(tbl, cb_content);
        hpdftbl_set_label_cb(tbl, cb_labels);
        hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
        HPDF_REAL xpos = hpdftbl_cm2dpi(1);
        HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
        HPDF_REAL width = hpdftbl_cm2dpi(12);
        HPDF_REAL height = 0;  // Calculate height automatically
        // Stroke the table to the page
        hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex06, FALSE)
```

17.10 tut_ex07.c

Expand cells over multiple columns and rows.
```
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    snprintf(buf, sizeof buf, "Content %02ix%02i", r, c);
#else
    snprintf(buf, sizeof buf, "Content %02zux%02zu", r, c);
#endif
    return buf;
}
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
    }
#else
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
#endif
    return buf;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex07(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    char *table_title = "tut_ex07: 7x5 table with row and colspans";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
```

```c
    hpdftbl_set_cellspan(tbl, 0, 0, 1, 3);
    hpdftbl_set_cellspan(tbl, 2, 2, 3, 3);
    hpdftbl_set_cellspan(tbl, 3, 0, 4, 1);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex07, FALSE)
```

17.11 tut_ex08.c

Adjust column width and expand cells over multiple columns and rows.

```c
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    snprintf(buf, sizeof buf, "Content %2ix%2i", r, c);
#else
    snprintf(buf, sizeof buf, "Content %zux%zu", r, c);
#endif
    return buf;
}
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
    }
#else
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
#endif
    return buf;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex08(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    char *table_title = "tut_ex08: 4x4 adjusting col width";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    hpdftbl_set_cellspan(tbl, 0, 0, 1, 3);
    hpdftbl_set_colwidth_percent(tbl, 0,40);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(17);
    HPDF_REAL height = 0;  // Calculate height automatically
```

```
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex08, FALSE)
```

17.12   tut_ex09.c

Adjusting font style with a callback.
```
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
_Bool
cb_style(void *tag, size_t r, size_t c, char *content, hpdf_text_style_t *style) {
    // Format the header row/column with a grey background and Helvetica font while the rest of the
    // table uses "Times Roman"
    if( 0==r || 0==c ) { // Headers
        style->font = HPDF_FF_HELVETICA_BOLD;
        style->fsize = 12;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_LIGHT_GRAY;
        if ( c > 0 )
            style->halign = CENTER;
        else
            style->halign = LEFT;
    } else { // Content
        style->font = HPDF_FF_TIMES;
        style->fsize = 11;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_WHITE;
        style->halign = CENTER;
    }
    return TRUE;
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    if( 0==r && 0==c ) return NULL;
    if( 0==c ) {
#if (defined _WIN32 || defined __WIN32__)
        snprintf(buf, sizeof buf, "Extra long Header %2ix%2i", r, c);
#else
        snprintf(buf, sizeof buf, "Extra long Header %zux%zu", r, c);
#endif
    } else if( 0==r ) {
#if (defined _WIN32 || defined __WIN32__)
        snprintf(buf, sizeof buf, "Header %2ix%2i", r, c);
#else
        snprintf(buf, sizeof buf, "Header %zux%zu", r, c);
#endif
    } else {
#if (defined _WIN32 || defined __WIN32__)
        snprintf(buf, sizeof buf, "Content %2ix%2i", r, c);
#else
        snprintf(buf, sizeof buf, "Content %zux%zu", r, c);
#endif
    }
    return buf;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex09(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_content_style_cb(tbl, cb_style);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex09, FALSE)
```

17.13 tut_ex10.c

Adjust column widths and add error handler.

```
#include "unit_test.inc.h"
void
create_table_ex10(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_colwidth_percent(tbl, 0, 30);
    hpdftbl_set_colwidth_percent(tbl, 1, 30);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex10, FALSE)
```

17.14 tut_ex11.c

Table with header row and error handler.

```
#include "unit_test.inc.h"
void
create_table_ex11(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex11, FALSE)
```

17.15 tut_ex12.c

Table with header row and error handler.

```
#include "unit_test.inc.h"
void
create_table_ex12(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex12, FALSE)
```

17.16 tut_ex13_1.c

Defining a table with a data structure for the table.
```c
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    if( 0==r )
        snprintf(buf, sizeof buf, "Header %02ix%02i", r, c);
    else
        snprintf(buf, sizeof buf, "Content %02ix%02i", r, c);
#else
    if( 0==r )
        snprintf(buf, sizeof buf, "Header %02zux%02zu", r, c);
    else
        snprintf(buf, sizeof buf, "Content %02zux%02zu", r, c);
#endif
    return buf;
}
static char *
cb_label(void *tag, size_t r, size_t c) {
    static char buf[32];
#if (defined _WIN32 || defined __WIN32__)
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
    }
#else
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
#endif
    return buf;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
hpdftbl_spec_t tbl_spec = {
        // Title and header flag
        .title=NULL, .use_header=TRUE,
        // Label and labelgrid flags
        .use_labels=FALSE, .use_labelgrid=FALSE,
        // Row and columns
        .rows=4, .cols=3,
        // xpos and ypos
        .xpos=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
        // width and height
        .width=hpdftbl_cm2dpi(15), .height=0,
        // Content and label callback
        .content_cb=cb_content, .label_cb=cb_label,
        // Style and table post creation callback
        .style_cb=NULL, .post_cb=NULL,
        // Pointer to optional cell specifications
        .cell_spec=NULL
};
void
create_table_ex13_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
}
TUTEX_MAIN(create_table_ex13_1, FALSE)
```

17.17 tut_ex13_2.c

Defining a table with a data structure for table and cells.
```c
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
```

```c
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char *cell_content[] =
            {"Mark Ericsen",
             "12 Sep 2021",
             "123 Downer Mews",
             "London",
             "NW2 HB3",
             "mark.p.ericsen@myfinemail.com",
             "+44734 354 184 56",
             "+44771 938 137 11"};
    if( 0==r && 0==c) return cell_content[0];
    else if (0==r && 3==c) return cell_content[1];
    else if (1==r && 0==c) return cell_content[2];
    else if (2==r && 0==c) return cell_content[3];
    else if (2==r && 3==c) return cell_content[4];
    else if (3==r && 0==c) return cell_content[5];
    else if (4==r && 0==c) return cell_content[6];
    else if (4==r && 2==c) return cell_content[7];
    else return NULL;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
hpdftbl_cell_spec_t cell_specs[] = {
        {.row=0, .col=0, .rowspan=1, .colspan=3,
                .label="Name:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=0, .col=3, .rowspan=1, .colspan=1,
                .label="Date:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=1, .col=0, .rowspan=1, .colspan=4,
                .label="Address:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=2, .col=0, .rowspan=1, .colspan=3,
                .label="City:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=2, .col=3, .rowspan=1, .colspan=1,
                .label="Zip:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=3, .col=0, .rowspan=1, .colspan=4,
                .label="E-mail:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=4, .col=0, .rowspan=1, .colspan=2,
                .label="Work-phone:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        {.row=4, .col=2, .rowspan=1, .colspan=2,
                .label="Mobile:",
                .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
        HPDFTBL_END_CELLSPECS
};
hpdftbl_spec_t tbl_spec = {
        // Title and header flag
        .title=NULL, .use_header=FALSE,
        // Label and labelgrid flags
        .use_labels=TRUE, .use_labelgrid=TRUE,
        // Row and columns
        .rows=5, .cols=4,
        // xpos and ypos
        .xpos=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
        // width and height
        .width=hpdftbl_cm2dpi(15), .height=0,
        // Content and label callback
        .content_cb=cb_content, .label_cb=0,
        // Style and table post creation callback
        .style_cb=NULL, .post_cb=NULL,
        // Pointer to optional cell specifications
        .cell_spec=cell_specs
};
void
create_table_ex13_2(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
}
TUTEX_MAIN(create_table_ex13_2, FALSE)
```

17.18 tut_ex14.c

Defining a table with widgets.

```c
#include "unit_test.inc.h"
```

```
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Device name:");
    } else if (0==r && 1==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else if (1==r && 0==c) {
        snprintf(buf, sizeof buf, "Battery strength:");
    } else if (1==r && 1==c) {
        snprintf(buf, sizeof buf, "Signal:");
    } else {
        return NULL;
    }
    return buf;
}
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_device_name(void *tag, size_t r, size_t c) {
    static char buf[32];
    snprintf(buf, sizeof buf, "IoT Device ABC123");
    return buf;
}
void
cb_draw_battery_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                       size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                       HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL segment_tot_width = width * 0.5;
    const HPDF_REAL segment_height = height / 3;
    const HPDF_REAL segment_xpos = xpos + 40;
    const HPDF_REAL segment_ypos = ypos + 4;
    const size_t num_segments = 10;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_GREEN;
    const double val_percent = 0.4;
    const _Bool val_text_hide = FALSE;
    hpdftbl_widget_segment_hbar(
            doc, page, segment_xpos, segment_ypos, segment_tot_width,
            segment_height, num_segments, on_color, val_percent, val_text_hide);
}
void
cb_draw_signal_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                      size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                      HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL wwidth = 35;
    const HPDF_REAL wheight = 20;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 4;
    const size_t num_segments = 5;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_RED;
    // This should be the real data retrieved from a DB (for example)
    const size_t num_on_segments = 3;
    hpdftbl_widget_strength_meter(doc, page, wxpos, wypos, wwidth, wheight,
                                  num_segments, on_color, num_on_segments);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex14(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex14: 2x2 table widget callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    // Use one label callback for the entire table
    hpdftbl_set_label_cb(tbl, cb_labels);
    // Name in top left corner
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_device_name);
    // Date in top right corner
    hpdftbl_set_cell_content_cb(tbl, 0, 1, cb_date);
```

```c
    // Draw battery strength
    hpdftbl_set_cell_canvas_cb(tbl, 1, 0, cb_draw_battery_widget);
    // Draw signal strength
    hpdftbl_set_cell_canvas_cb(tbl, 1, 1, cb_draw_signal_widget);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex14, FALSE)
```

17.19   tut_ex15.c

Defining a table with zebra lines.
```c
#include "unit_test.inc.h"
void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    //hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_zebra(tbl, TRUE, 0);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex15, FALSE)
```

17.20   tut_ex15_1.c

Defining a table with zebra lines and different phase.
```c
#include "unit_test.inc.h"
void
create_table_ex15_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_zebra(tbl, TRUE, 1);
    hpdftbl_set_inner_hgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY,LINE_SOLID );
    hpdftbl_set_inner_tgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY,LINE_DOT1 );
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex15_1, FALSE)
```

17.21   tut_ex20.c

Defining a table and adjusting the gridlines.
```c
#include "unit_test.inc.h"
void
create_table_ex20(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 5;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
```

```
        content_t content, labels;
        setup_dummy_content_label(&content, &labels, num_rows, num_cols);
        hpdftbl_set_content(tbl, content);
        hpdftbl_set_labels(tbl, labels);

        hpdftbl_use_labels(tbl, FALSE);
        hpdftbl_use_labelgrid(tbl, TRUE);
        hpdftbl_use_header(tbl, FALSE);
        hpdftbl_set_inner_vgrid_style(tbl, 0.7, HPDF_COLOR_DARK_GRAY, LINE_SOLID);
        hpdftbl_set_inner_hgrid_style(tbl, 0.8, HPDF_COLOR_GRAY, LINE_DOT1);
        hpdftbl_set_inner_tgrid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
        hpdftbl_set_outer_grid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
        HPDF_REAL xpos = hpdftbl_cm2dpi(1);
        HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
        HPDF_REAL width = hpdftbl_cm2dpi(10);
        HPDF_REAL height = 0;  // Calculate height automatically
        // Stroke the table to the page
        hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex20, FALSE)
```

17.22 tut_ex30.c

Defining a table using dynamic callbacks
```
#include "dlfcn.h"
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#pragma GCC diagnostic ignored "-Wunused-function"
#endif
char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    snprintf(buf, sizeof buf, "Content %02zu x %02zu", r, c);
    return buf;
}
char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date created:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
    return buf;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex30(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex30: Table with dynamic callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_dyncb(tbl, "cb_content");
    hpdftbl_set_label_dyncb(tbl, "cb_labels");
    hpdftbl_set_cell_content_dyncb(tbl, 0, 0, "cb_date");
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0;  // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex30, FALSE)
```

17.23   tut_ex40.c

Example of importing a table from a serialized json file.

See also

[hpdftbl_dump()](#)

```c
#include <yaml.h>
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#pragma GCC diagnostic ignored "-Wunused-function"
#pragma GCC diagnostic ignored "-Wunused-variable"
#endif
#define FROM_JSON 1
void
create_table_ex40(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    hpdftbl_t tbl=calloc(1, sizeof(struct hpdftbl));
#if FROM_JSON == 1
    if(0 == hpdftbl_load(tbl, mkfullpath("tut_ex40.json"))  ) {
        hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
    }
#else
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    tbl = hpdftbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_cellspan(tbl,0,0,1,2);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
    hpdftbl_dump(tbl, "out/tut_ex40.json");
#endif
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
TUTEX_MAIN(create_table_ex40, FALSE)
```

17.24   tut_ex41.c

Example of importing a table and theme from a serialized representation.

See also

[hpdftbl_load()](#), [hpdftbl_theme_load()](#)

```c
#include <yaml.h>
#include "unit_test.inc.h"
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#pragma GCC diagnostic ignored "-Wunused-function"
#pragma GCC diagnostic ignored "-Wunused-variable"
#endif
#define FROM_JSON 0
void
create_table_ex41(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
#if FROM_JSON == 0
    hpdftbl_t tbl = calloc(1, sizeof (struct hpdftbl));
```

```c
    hpdftbl_theme_t theme;
    if(0 == hpdftbl_load(tbl, mkfullpath("tut_ex41.json"))   ) {
        if(0 == hpdftbl_theme_load(&theme, mkfullpath("tut41_theme.json"))) {
            hpdftbl_apply_theme(tbl, &theme);
            hpdftbl_stroke_pos(pdf_doc, pdf_page, tbl);
        } else {
            fprintf(stderr,"%s\n","Failed to load 'tests/default_theme.json'\n");
            exit(1);
        }
    } else {
        fprintf(stderr,"%s\n","Failed to load 'tests/tut_ex41.json'\n");
        exit(1);
    }
#else
    const size_t num_rows = 4;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_content_label(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_set_zebra(tbl, TRUE, 0);
    hpdftbl_set_cellspan(tbl,0,0,1,2);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(8);
    HPDF_REAL height = 0;  // Calculate height automatically
    hpdftbl_theme_t *theme = hpdftbl_get_default_theme();
    hpdftbl_get_theme(tbl,theme);
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
    hpdftbl_dump(tbl, "out/tut_ex41.json");
    hpdftbl_theme_dump(theme,"out/tut41_theme.json");
#endif
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
TUTEX_MAIN(create_table_ex41, FALSE)
```

17.25   tests/tut_ex40.json

An output example from hpdftbl_dump() that shows a serialized table. This can later be import to a table structure with the use of hpdftbl_load()

```json
{
"version": 1,
"table": {
  "tag": "",
  "rows": 2,
  "cols": 2,
  "posx": 28.34645653,
  "posy": 813.54333496,
  "height": 57.00000000,
  "minrowheight": 0.00000000,
  "width": 141.73228455,
  "bottom_vmargin_factor": 0.50000000,
  "title_txt": "",
  "anchor_is_top_left": true,
  "use_header_row": false,
  "use_cell_labels": true,
  "use_label_grid_style": true,
  "use_zebra": false,
  "zebra_phase": 0,
  "zebra_color1": [1.00000, 1.00000, 1.00000],
  "zebra_color2": [0.95000, 0.95000, 0.95000],
  "outer_grid": {
    "width": 1.00000000,
    "color": [0.20000, 0.20000, 0.20000],
    "dashstyle": 0
  },
  "inner_vgrid": {
    "width": 0.69999999,
    "color": [0.50000, 0.50000, 0.50000],
    "dashstyle": 0
  },
  "inner_hgrid": {
    "width": 0.69999999,
    "color": [0.50000, 0.50000, 0.50000],
    "dashstyle": 0
  },
```

```
"inner_tgrid": {
  "width": 0.69999999,
  "color": [0.50000, 0.50000, 0.50000],
  "dashstyle": 0
},
"content_style": {
  "font": "Courier",
  "fsize": 10.00000000,
  "color": [0.20000, 0.20000, 0.20000],
  "background": [1.00000, 1.00000, 1.00000],
  "halign": 0
},
"title_style": {
  "font": "Helvetica-Bold",
  "fsize": 11.00000000,
  "color": [0.00000, 0.00000, 0.00000],
  "background": [0.90000, 0.90000, 0.90000],
  "halign": 0
},
"header_style": {
  "font": "Helvetica-Bold",
  "fsize": 10.00000000,
  "color": [0.00000, 0.00000, 0.00000],
  "background": [0.90000, 0.90000, 0.97000],
  "halign": 1
},
"label_style": {
  "font": "Times-Italic",
  "fsize": 9.00000000,
  "color": [0.40000, 0.40000, 0.40000],
  "background": [1.00000, 1.00000, 1.00000],
  "halign": 0
},
"col_width_percent": [
  50.000000, 50.000000
],
"label_dyncb": "",
"content_dyncb": "",
"content_style_dyncb": "",
"canvas_dyncb": "",
"post_dyncb": "",
"cells": [
  {
    "row": 0,
    "col": 0,
    "label": "Label 0:",
    "content": "Content 0",
    "colspan": 2,
    "rowspan": 1,
    "height": 28.50000000,
    "width": 141.73228455,
    "delta_x": 0.00000000,
    "delta_y": 28.50000000,
    "textwidth": 0.00000000,
    "content_dyncb": "",
    "label_dyncb": "",
    "content_style_dyncb": "",
    "canvas_dyncb": "",
    "content_style": {
      "font": "",
      "fsize": 0.00000000,
      "color": [0.00000, 0.00000, 0.00000],
      "background": [0.00000, 0.00000, 0.00000],
      "halign": 0
    }
  },
  {
    "row": 0,
    "col": 1,
    "label": "Label 1:",
    "content": "Content 1",
    "colspan": 1,
    "rowspan": 1,
    "height": 28.50000000,
    "width": 70.86614227,
    "delta_x": 70.86614227,
    "delta_y": 28.50000000,
    "textwidth": 0.00000000,
    "content_dyncb": "",
    "label_dyncb": "",
    "content_style_dyncb": "",
    "canvas_dyncb": "",
    "parent": {
      "row": 0,
      "col": 0
    },
    "content_style": {
```

```
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    },
    {
      "row": 1,
      "col": 0,
      "label": "Label 2:",
      "content": "Content 2",
      "colspan": 0,
      "rowspan": 0,
      "height": 28.50000000,
      "width": 70.86614227,
      "delta_x": 0.00000000,
      "delta_y": 0.00000000,
      "textwidth": 0.00000000,
      "content_dyncb": "",
      "label_dyncb": "",
      "content_style_dyncb": "",
      "canvas_dyncb": "",
      "content_style": {
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    },
    {
      "row": 1,
      "col": 1,
      "label": "Label 3:",
      "content": "Content 3",
      "colspan": 0,
      "rowspan": 0,
      "height": 28.50000000,
      "width": 70.86614227,
      "delta_x": 70.86614227,
      "delta_y": 0.00000000,
      "textwidth": 0.00000000,
      "content_dyncb": "",
      "label_dyncb": "",
      "content_style_dyncb": "",
      "canvas_dyncb": "",
      "content_style": {
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    }
    ]
  }
}
```

17.26 tests/tut_ex41.json

An output example from hpdftbl_dump() as well as hpdftbl_theme_dump() that shows a serialized table and theme. This can later be import to a table structure with the use of hpdftbl_load() and hpdftbl_theme_import()

```
{
"version": 1,
"table": {
  "tag": "",
  "rows": 4,
  "cols": 2,
  "posx": 28.34645653,
  "posy": 813.54333496,
  "height": 114.00000000,
  "minrowheight": 0.00000000,
  "width": 226.77165222,
  "bottom_vmargin_factor": 0.50000000,
  "title_txt": "",
  "anchor_is_top_left": true,
  "use_header_row": false,
  "use_cell_labels": true,
  "use_label_grid_style": true,
  "use_zebra": true,
```

```
"zebra_phase": 0,
"zebra_color1": [1.00000, 1.00000, 1.00000],
"zebra_color2": [0.95000, 0.95000, 0.95000],
"outer_grid": {
  "width": 1.00000000,
  "color": [0.20000, 0.20000, 0.20000],
  "dashstyle": 0
},
"inner_vgrid": {
  "width": 0.69999999,
  "color": [0.50000, 0.50000, 0.50000],
  "dashstyle": 0
},
"inner_hgrid": {
  "width": 0.69999999,
  "color": [0.50000, 0.50000, 0.50000],
  "dashstyle": 0
},
"inner_tgrid": {
  "width": 0.69999999,
  "color": [0.50000, 0.50000, 0.50000],
  "dashstyle": 0
},
"content_style": {
  "font": "Courier",
  "fsize": 10.00000000,
  "color": [0.20000, 0.20000, 0.20000],
  "background": [1.00000, 1.00000, 1.00000],
  "halign": 0
},
"title_style": {
  "font": "Helvetica-Bold",
  "fsize": 11.00000000,
  "color": [0.00000, 0.00000, 0.00000],
  "background": [0.90000, 0.90000, 0.90000],
  "halign": 0
},
"header_style": {
  "font": "Helvetica-Bold",
  "fsize": 10.00000000,
  "color": [0.00000, 0.00000, 0.00000],
  "background": [0.90000, 0.90000, 0.97000],
  "halign": 1
},
"label_style": {
  "font": "Times-Italic",
  "fsize": 9.00000000,
  "color": [0.40000, 0.40000, 0.40000],
  "background": [1.00000, 1.00000, 1.00000],
  "halign": 0
},
"col_width_percent": [
  50.000000, 50.000000
],
"label_dyncb": "",
"content_dyncb": "",
"content_style_dyncb": "",
"canvas_dyncb": "",
"post_dyncb": "",
"cells": [
  {
    "row": 0,
    "col": 0,
    "label": "Label 0:",
    "content": "Content 0",
    "colspan": 2,
    "rowspan": 1,
    "height": 28.50000000,
    "width": 226.77165222,
    "delta_x": 0.00000000,
    "delta_y": 85.50000000,
    "textwidth": 0.00000000,
    "content_dyncb": "",
    "label_dyncb": "",
    "content_style_dyncb": "",
    "canvas_dyncb": "",
    "content_style": {
      "font": "",
      "fsize": 0.00000000,
      "color": [0.00000, 0.00000, 0.00000],
      "background": [0.00000, 0.00000, 0.00000],
      "halign": 0
    }
  },
  {
    "row": 0,
    "col": 1,
```

```
        "label": "Label 1:",
        "content": "Content 1",
        "colspan": 1,
        "rowspan": 1,
        "height": 28.50000000,
        "width": 113.38582611,
        "delta_x": 113.38582611,
        "delta_y": 85.50000000,
        "textwidth": 0.00000000,
        "content_dyncb": "",
        "label_dyncb": "",
        "content_style_dyncb": "",
        "canvas_dyncb": "",
        "parent": {
          "row": 0,
          "col": 0
        },
        "content_style": {
          "font": "",
          "fsize": 0.00000000,
          "color": [0.00000, 0.00000, 0.00000],
          "background": [0.00000, 0.00000, 0.00000],
          "halign": 0
        }
      },
      {
        "row": 1,
        "col": 0,
        "label": "Label 2:",
        "content": "Content 2",
        "colspan": 0,
        "rowspan": 0,
        "height": 28.50000000,
        "width": 113.38582611,
        "delta_x": 0.00000000,
        "delta_y": 57.00000000,
        "textwidth": 0.00000000,
        "content_dyncb": "",
        "label_dyncb": "",
        "content_style_dyncb": "",
        "canvas_dyncb": "",
        "content_style": {
          "font": "",
          "fsize": 0.00000000,
          "color": [0.00000, 0.00000, 0.00000],
          "background": [0.00000, 0.00000, 0.00000],
          "halign": 0
        }
      },
      {
        "row": 1,
        "col": 1,
        "label": "Label 3:",
        "content": "Content 3",
        "colspan": 0,
        "rowspan": 0,
        "height": 28.50000000,
        "width": 113.38582611,
        "delta_x": 113.38582611,
        "delta_y": 57.00000000,
        "textwidth": 0.00000000,
        "content_dyncb": "",
        "label_dyncb": "",
        "content_style_dyncb": "",
        "canvas_dyncb": "",
        "content_style": {
          "font": "",
          "fsize": 0.00000000,
          "color": [0.00000, 0.00000, 0.00000],
          "background": [0.00000, 0.00000, 0.00000],
          "halign": 0
        }
      },
      {
        "row": 2,
        "col": 0,
        "label": "Label 4:",
        "content": "Content 4",
        "colspan": 0,
        "rowspan": 0,
        "height": 28.50000000,
        "width": 113.38582611,
        "delta_x": 0.00000000,
        "delta_y": 28.50000000,
        "textwidth": 0.00000000,
        "content_dyncb": "",
        "label_dyncb": "",
```

```
      "content_style_dyncb": "",
      "canvas_dyncb": "",
      "content_style": {
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    },
    {
      "row": 2,
      "col": 1,
      "label": "Label 5:",
      "content": "Content 5",
      "colspan": 0,
      "rowspan": 0,
      "height": 28.50000000,
      "width": 113.38582611,
      "delta_x": 113.38582611,
      "delta_y": 28.50000000,
      "textwidth": 0.00000000,
      "content_dyncb": "",
      "label_dyncb": "",
      "content_style_dyncb": "",
      "canvas_dyncb": "",
      "content_style": {
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    },
    {
      "row": 3,
      "col": 0,
      "label": "Label 6:",
      "content": "Content 6",
      "colspan": 0,
      "rowspan": 0,
      "height": 28.50000000,
      "width": 113.38582611,
      "delta_x": 0.00000000,
      "delta_y": 0.00000000,
      "textwidth": 0.00000000,
      "content_dyncb": "",
      "label_dyncb": "",
      "content_style_dyncb": "",
      "canvas_dyncb": "",
      "content_style": {
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    },
    {
      "row": 3,
      "col": 1,
      "label": "Label 7:",
      "content": "Content 7",
      "colspan": 0,
      "rowspan": 0,
      "height": 28.50000000,
      "width": 113.38582611,
      "delta_x": 113.38582611,
      "delta_y": 0.00000000,
      "textwidth": 0.00000000,
      "content_dyncb": "",
      "label_dyncb": "",
      "content_style_dyncb": "",
      "canvas_dyncb": "",
      "content_style": {
        "font": "",
        "fsize": 0.00000000,
        "color": [0.00000, 0.00000, 0.00000],
        "background": [0.00000, 0.00000, 0.00000],
        "halign": 0
      }
    }
    ]
  }
}
```

Index