libhpdftbl

Generated on Thu Apr 28 2022 06:51:24 for libhpdftbl by Doxygen 1.9.3

Thu Apr 28 2022 06:51:24

Chapter 1

Introduction to hpdftbl

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.

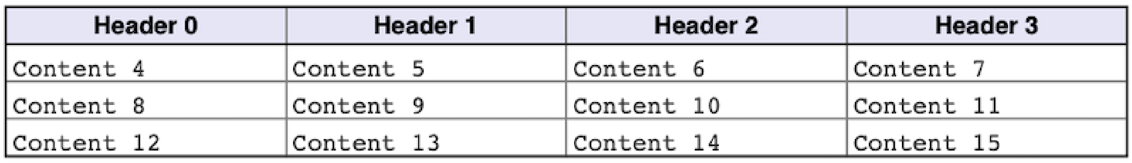
1.2 Features

- Supports both C/C++
- Suports both OSX/Linux builds and theire different dynamic library variants
- Fully supports UTF-8 with automatic conversion to PDF character encoding
- Supports multple 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
- Options to use labels in table cell to create forms
- 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
- Both dynamic and static library provided
- Last but not least; extensive documentation and almost guaranteed to be bug free after beeing tested in production for over 7 years!

1.3 Some Examples

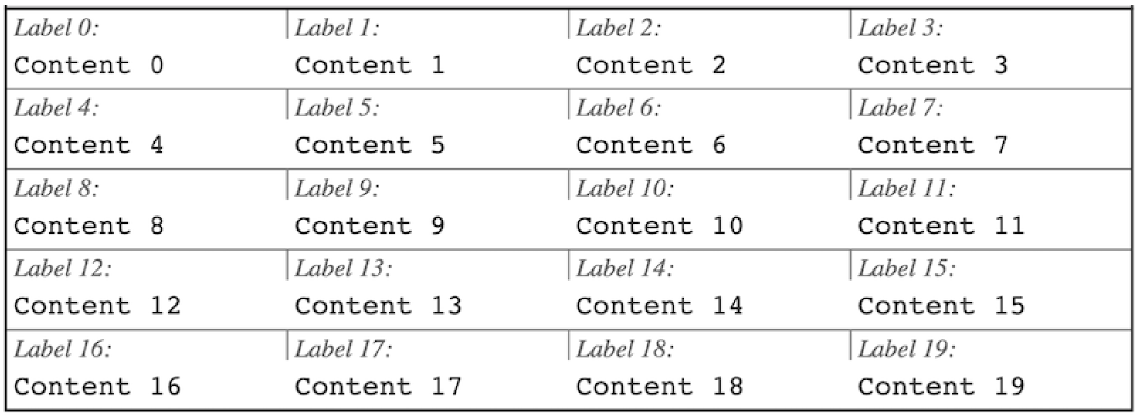
Note: All code examples can be found in the "`examples/`" directory and are thoroughly introduced over the following chapters.

1.3.1 Example 1 - Plain table with cell labels



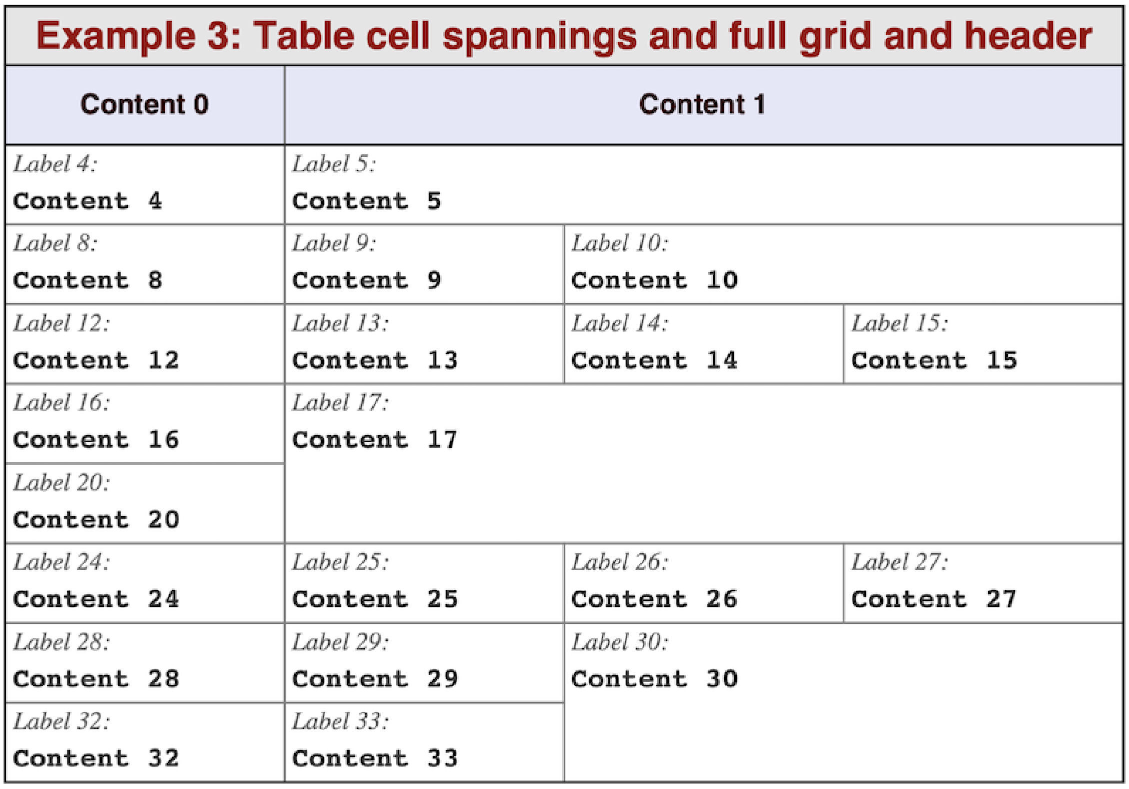
Header 0	Header 1	Header 2	Header 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15

1.3.2 Example 2 - Table with cell labels



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

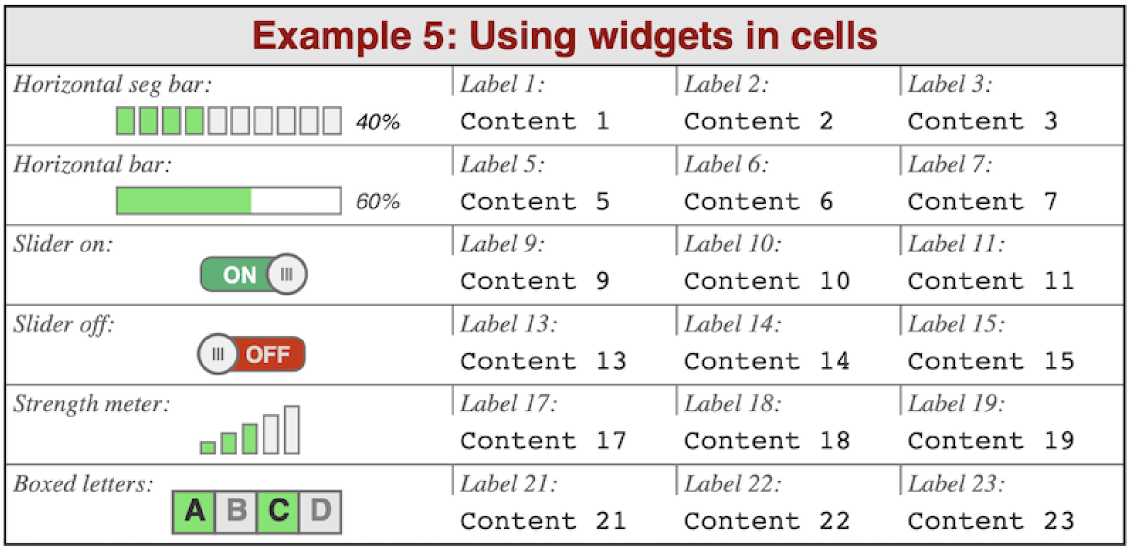
1.3.3 Example 2 - Plain table with row/column spanning and table title



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



Example 5: Using widgets in cells

Horizontal seg bar: 🟩🟩🟩🟩⬜⬜⬜⬜⬜⬜ 40%	*Label 1:* `Content 1`	*Label 2:* `Content 2`	*Label 3:* `Content 3`
Horizontal bar: 60%	*Label 5:* `Content 5`	*Label 6:* `Content 6`	*Label 7:* `Content 7`
Slider on: ON	*Label 9:* `Content 9`	*Label 10:* `Content 10`	*Label 11:* `Content 11`
Slider off: 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: A B C D	*Label 21:* `Content 21`	*Label 22:* `Content 22`	*Label 23:* `Content 23`

Chapter 2

Building the library

2.1 The short version

2.1.1 Compiling the tar ball

If you downloaded the tar-ball then it should be trivial to build and install if you have the necessary pre-requisites. Just download the tar-ball and do the standard spell:

```
$ tar xzf libhpdftbl-1.0.0.tar.gz
$ cd libhpdf-1.0.0
$ ./configure && make
$ make install
```

If you miss any library the `configure` process will discover this and tell you what you need to install. This would otherwise compile and install the library in `/usr/local` subtree. It wil build and install both a static and dynamic library.

Depending on your system this might also be available as a pre-built package for you to install directly via perhaps `apt`on Linux or `brew` on OSX.

2.1.2 Compiling after cloning the git repo

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

1. A complete set of GNU compiler chain (or on OSX clang)
2. An installation of the autotools (autoconf, automake, libtool)
3. An installation of Doxygen (to generate documentation)

If these three pre-requisites are installed then the build environment is bootstrapped by running

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

and then continue to compile

```
$ make
```

and (optionally) install the library

```
$ make install
```

2.2 Pre-requisites

Note

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

There are two external libraries required to rebuild libhpdftbl and more importantly use the library with an actual application and these are:

1. **libhpdf** - The Haru PDF library. On OSX this is most easly 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 really old versions of OSX this was not the case.)∗

2.3 Different versions of iconv on OSX

Unfortunately there are two main versions of `libiconv` readily available for OSX which are incompatible as one uses the prefix "`iconv_∗`" and the other "`libiconv_∗`" on its exported functions. Compiling `libhpdftbl` requires the first of these which is the prevelant version and the default on both OSX and Linux.

This is almost exclusivly 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.

2.3.1 OSX native libiconv

After installing `xcode` command line tools on OSX you can assume that a library called `/usr/lib/iconv.dylib` is available. However, if you actually try to list this library in `/usr/lib` you will not find it! Still, if you link your code 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 librares are rolled-up in a dynamic link shared cache for each SDK version. The tool chain (e.g. `gcc`) 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* `∗.tbd` (tbd stands for "text based description") which are much smaller text files with some 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 at
`/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/iconv.h`

2.3.2 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)

2.3.3 Troubleshooting OSX <tt>libiconv</tt>

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

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

   The "`2> /dev/null`" makes sure you don't get a lot of noise "permission denied"
2. Find out the SDK path that is actively used

   ```
   $> xcrun --show-sdk-path
   ```
3. Check you `PATH` variable

   ```
   $> echo $PATH
   ```

2.4 Building the library from source

There are two levels of rebuilding the library

1. Using a build environment to rebuild the library
2. Rebuilding from a cloned repo and rebuild the build environment

2.4.1 Rebuilding using a existing build environment

Rebuilding the library using a pre-configured build environment only requires `gcc` and `make` together with the standard C/C++ libraries to be installed.

The library source with suitable build-environments are distributed as a tar-ball

1. libhpdf-src-x.y.z.tar.gz

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

```
$> ./configure && make
... (output from the configuration and build omitted) ...
```

Note

: The git repo do not have a build environment setup.

2.4.2 Rebuilding from the cloned repo

Rebuilding from the cloned repo requires the GNU autotools tool-chain to be installed. Since it is completely out of the scope to decribe the intricacies of the GNU autotools we will only show what to do assuming this tool chain have been installed.

To simplify the potentially painful bootstrap of creating a full autotools environment a utility script that does this is provided in the form of "`scripts/bootstrap.sh`". After cloning the repo run (from the `libhpdftbl` 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: --------------------------------------------------------------------------------
```

The final step you need to do is compile the library as so

```
$> make
```

The simplest way to verify that everything works is to execute one of the example programs (in the `examples/` directory) as so:

```
$> ./examples/example01
Stroking 5 examples.
Sending to file "/tmp/example01.pdf" ...
Done.
```

If you would like to install the library make the install target

```
$> make install
```

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

2.5 Some notes on Windows build

The source files are suitable augmented to also compile on MS Windows with selective defines. However, since I have no access to a Windows system to verify the workings this is left as an exercise to the reader.

2.6 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 setup 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 you to the basic usage of the `hpdftbl` library. We will start simple and work us all the way to complex tables and exaplin 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 a PDF page infrastructure

Before we start creating a table we need to setup 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 simple setup for all our examples. We will create a document in A4 size that have one page. For this we use a few utility functions and our `main()` will always have the following structure:

```
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, TRUE);
    create_table_<NAME_OF_EXAMPLE>(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
```

In the Appendix you can find the full code for the setup and troke function. They are very basic and follows the standard hpdf library methoddology. The `setup_hpdf()` creates a new document and a A4 page and the `stroke_pdfdoc()` strokes the document to the given output file.

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 compile 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 manul.

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

First we construct a table handle for a 2x2 table

```
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
- Most other table function will refer to this handle and we will always use the varaiable 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 alwyas 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");
```

Note

You can ignore the `NULL` argument for now (it will be explained shortly).

Here we note that:

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

Now its time to size and position the 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)

The native coordiante 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. We positionin the top left of the table *1cm* below and *1cm* to the right of the top left corner of the paper and make the table *5cm* 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

```
hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
```

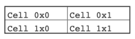
and we are done!

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

```
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");
    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);
}
```

The generated table is shown in **Figure 1.** (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. If we add the grid to the page and show the uppper left area of the paper with the grid we can view its positioning in the grid as shown in **Figure 2.**

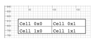


Figure 2: *Your first table in the page coordinate system showing the upper left part of the paper.*

Since this is an A4 page it will have a height of roughly 841 points or 29.7cm

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 suppply 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 ecah 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:

```
typedef char **content_t;
void setup_dummy_data(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++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
```

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!

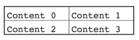
We could then augment example 01 using this more efficient way to specify data as so:

```
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_data(&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(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



Content 0	Content 1
Content 2	Content 3

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 effiecient 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 decribed 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.

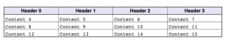
3.4 Adding a header row

While it is possible (as discussed in section ??) to manually adjust the font, size, style, background etc. on each cell individually there is a convinient 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

```
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_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    content_t content;
    setup_dummy_data(&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

    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" in the first row (for details see tut_ex02_1.c)

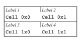


Header 0	Header 1	Header 2	Header 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15

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 dataform. An example of this is shown in **Figure 4.** below.

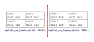


Label 1 Cell 0x0	*Label 2* Cell 0x1
Label 3 Cell 1x0	*Label 4* Cell 1x1

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 lable height as was shown in **Figure 4.** above. This option is specified with `hpdftbl_use_labelgrid()`. By defaullt the left border is from top to bottom. The differenceies between the two variants is shown in **Figure 5.** below.



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

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

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

We therefore start by amending our dummy data creation function to also create the data for the labels. It will now look like this:

```
typedef char **content_t;
void
setup_dummy_data(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

```
setup_dummy_data(&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.

```
void
create_table_ex04(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, labels;
    setup_dummy_data(&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
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

tut_ex04.c

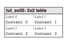
3.6 Adding a table title

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 it's own row which isn't part of the counting if the normal columns.



tut_ex05: 2x2 table	
Label 0 Content 0	*Label 1* Content 1
Label 2 Content 2	*Label 3* Content 3

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

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

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 different granularities. 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()`

1. Adjust one entire row in using `hpdftbl_set_row_content_style()`

1. 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

Adjusting the layout of the table

The table can be modified both by adjusting the width of columns as well as 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 col 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 these cell-spannings:

```
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.**

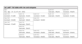


Figure 8: ∗Having cells spanning multiple rows and columns. tut_ex07.c∗

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%.

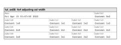


Figure 9: ∗Adjusting width of first columns. tut_ex08.c ∗

Chapter 5

Content and label 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 table library for each cell and returns a string which is used as tne data to be displayed. The signature for a cell 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);
```

To understand this lets start with a callback function that follows this signature.

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

The parameters in the callback are

1. `**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 retrives data for this particular table. The `tag` for a table is specified with the `hpdftbl_set_tag()` function. When the callback is made this table tag is provided as the first argument.
2. `**row**`: The cell row
3. `**col**`: The cell column

It is possible to specify a callback to adjust content, style, and labels. A callback function can be specfied either for both the entire table as well as individual cells. The API to specify these callbacks are:

1. `hpdftbl_set_content_cb()`:
   Specify a content callback for the entire table.

2. `hpdftbl_set_content_style_cb()`:
   Specify a style callback for the entire table.

3. `hpdftbl_set_label_cb()`:
   Specify a label callback for the entire table.

4. `hpdftbl_set_cell_content_cb()`:
   Specify callback for an individual cell. A cell callback will override a potential table callback.

5. `hpdftbl_set_cell_content_style_cb()`:
   Specify a style callback for an individual cell. A cell callback will override a potential table callback.

6. `hpdftbl_set_canvas_cb()`: This is an advanced callback to allow for low level painting directly on the canvas that is the cell area. The 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**. When a content string is added in the table it is added as a copy of the string pointed to by the returned string pointer from the callback function. It is therefore perfectly possible to have a static allocated buffer in the callback function that is used to construct the content. When the table is destroyd using `hpdftbl_destroy()` all used memory will be freed.

5.2 A content callback example

Let's now construct a simple example 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 three callback functions we need

```
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;
}
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 a callback comes.

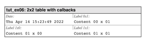
For the table structure we will re-use our previous example and create a 2x2 table and we get the following table creation code:

```
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 calbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_content_cb(tbl, cb_content);
    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

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

(tut_ex05.c)

Running this example gives the result shown in **Figure 7.** below



tut_ex06: 2x2 table with calbacks	
Date: Thu Apr 14 15:23:49 2022	*Label 0x1:* Content 00 x 01
Label 1x0: Content 01 x 00	*Label 1x1:* Content 01 x 01

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

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 examplified by a call to `hpdftbl_set_colwidth_percent()`

```
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 longwinding error handling if one where to check every soingle library call. Instead there is the option of installing an error handler that would be called in the eent of an error.

The table error handle has the signature

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

Where the arguments are

1. `tbl` The table in where the error happened. **Note** This might be `NULL´ since not all errors happen within the context of a table 2.r,c`The row and column if the error happens in a specified cell, otherwise these will be (-1,-1) `3.`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:

```
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(1);
}
```

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 convinience in the librry 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 thr particular application they are to be used in.

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 rememberd 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`

6.1 Translating HPDF error codes

The standard error handler for the plain HPDF library is specified when a new document is created, for example as'

```
...
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

```
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 convinience function to translate the Haru library error code into a human readable string. This function is

```
const char *
hpdftbl_hpdf_get_errstr(const HPDF_STATUS err_code)
```

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

6.2 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, 40);
    hpdftbl_set_colwidth_percent(tbl, 1, 70);
    content_t content;
    setup_dummy_data(&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. When this code is executed the following will be printed to standard out and the process will be stopped.

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

Chapter 7

Style and font 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 styld can be adjusted both forthe entire table at once and alse 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 conten 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 Specifying 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 COLOR_DARK_RED      (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
#define COLOR_RED           (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
#define COLOR_LIGHT_GREEN   (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
#define COLOR_GREEN         (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
#define COLOR_DARK_GRAY     (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
#define COLOR_LIGHT_GRAY    (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
#define COLOR_GRAY          (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
#define COLOR_SILVER        (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
#define COLOR_LIGHT_BLUE    (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
#define COLOR_BLUE          (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
#define COLOR_WHITE         (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
#define 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, COLOR_BLACK, COLOR_WHITE);
...
```

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

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

which will allow the easier specification of color such as

```
HPDF_RGBColor color_saddle_brown = HPDF_COLOR_FROMRGB(139,69,19);
```

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 higligthning 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

```
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 setup style callbacks are

```
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 setup 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

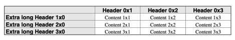
An example of a callback function to set a background color for a header row/column for a table could for example be done as follows

```
_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 = COLOR_BLACK;
        style->background = 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 = COLOR_BLACK;
        style->background = COLOR_WHITE;
        style->halign = CENTER;
    }
    return TRUE;
}
```

and the table setup code can then be written as

```
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.



	Header 0x1	Header 0x2	Header 0x3
Extra long Header 1x0	Content 1x1	Content 1x2	Content 1x3
Extra long Header 2x0	Content 2x1	Content 2x2	Content 2x3
Extra long Header 3x0	Content 3x1	Content 3x2	Content 3x3

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

7.3 Using style themes

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. The data saved in a theme is defined by the structure hpdftbl_theme with the following definition and members

```
typedef struct hpdftbl_theme {
    hpdf_text_style_t *content_style;
    hpdf_text_style_t *label_style;
    hpdf_text_style_t *header_style;
    hpdf_text_style_t *title_style;
    hpdf_border_style_t *inner_border;
    hpdf_border_style_t *outer_border;
    _Bool use_labels;
    _Bool use_label_grid_style;
    _Bool use_header_row;
} hpdftbl_theme_t;
```

This allow the setting of all main font/style setting in one go. This strcture 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. The functions to work with a theme are:

```
// Apply the given theme to a table
int
hpdftbl_apply_theme(hpdftbl_t t, hpdftbl_theme_t *theme);
// Get the default theme into a new allocated structure
hpdftbl_theme_t *
hpdftbl_get_default_theme(void);
// Destroy the memory used by a theme
int
hpdftbl_destroy_theme(hpdftbl_theme_t *theme);
```

Note

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

Table 2: *Default table structure parameters.*

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

Table 3: *Default border parameters.*

Note

There is currently no support for serializing a theme to/from a file.

Chapter 8

Tables layout from data

So far we have constructed the layout of table by issuing API calls per table to setup, 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 insted 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 bed defined entirely in a database and makes it possible to adjust tha table as the data changes without ever updating the code (or recompile).

8.1 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 as was discussed in secion ??. 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 (as described in section ??)

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;
```

8.2 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,
        // 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
};
```

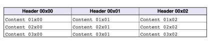
Note

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 to what we seen before 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!



Header 00x00	Header 00x01	Header 00x02
Content 01x00	Content 01x01	Content 01x02
Content 02x00	Content 02x01	Content 02x02
Content 03x00	Content 03x01	Content 03x02

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

8.3 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 mans model-view-controller where the table structure is completely separate from the

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

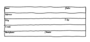


Figure 14: *Sketch of table to be designed*

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 or cells span multiple columns.

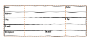


Figure 15: *Sketch of table to be designed with 5x4 table overlayed*

To start we setup 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 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="Workphone:",
         .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 a 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 13.2**

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 to far. Instead we will just use some fake static dummmy 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 9

Widgets

9.1 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.

9.1.1 1. Segmented horizontal bar example

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

9.1.2 2. Horizontal bar example

Basic horizontal bar

9.1.3 3. Signal strength meter example

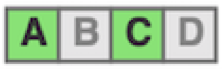
A widget indicate a signal strength in similar fashion as the signal strength meter on a phone.

9.1.4 4. Radio sliding button example

Radio button/Slider with different on/off

9.1.5 5. Boxed letters example

Highlight zero or more letters



9.2 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.

9.2.1 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)
```

9.2.2 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)
```

9.2.3 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)
```

9.2.4 Radio sliding button defining function

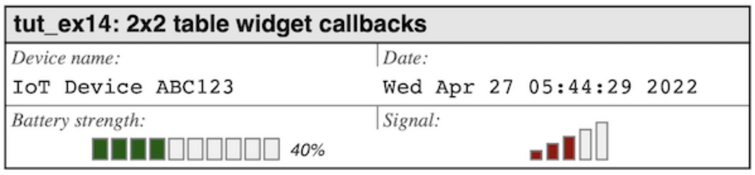
```
void
hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
                            HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, _Bool state)
```

9.2.5 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)
```

9.3 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



tut_ex14: 2x2 table widget callbacks

Device name: IoT Device ABC123	*Date:* Wed Apr 27 05:44:29 2022
Battery strength: 40%	*Signal:*

Figure 9.1 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 = 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 = 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 10

HPDFTBL API Overview

10.1 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_stroke_from_data() *Construct and stroke a table defined as a data 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 betwen top left and bottom left corner.*
- hpdftbl_get_anchor_top_left() *Get the current achor point of table.*

10.2 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.*

10.3 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 tabl.e*
- 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.*

10.4 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_cellspan() ∗Define a cell to span multiple rows and columns."
- hpdftbl_clear_spanning() *Remove all previous set cell spanning.*

10.5 Table style modifying functions

These function are all about look an 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_border() *Set style of the table outer border.*
- hpdftbl_set_inner_border() *Set the style of table inner borders.*
- 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 stle for specified cell. This overrides andy style on the table level.*
- hpdftbl_set_title_style *Set the style for the table title.*

10.6 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.*

10.7 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.*

10.8 Text encoding

- hpdftbl_set_text_encoding() *Specify text encodation to use.*
- hpdftbl_encoding_text_out() *Stroke a text with current encoding.*

10.9 Misc utility function

- HPDF_RoundedCornerRectangle() *Draw a rectanle 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 11

Data Structure Index

11.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 12

File Index

12.1 File List

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

Chapter 13

Data Structure Documentation

13.1 border_style Struct Reference

Specification for table borders.

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

Data Fields

- HPDF_REAL width
- HPDF_RGBColor color
- hpdftbl_line_style_t line_style

13.1.1 Detailed Description

Specification for table borders.

Contains line properties used when stroking a border line

13.1.2 Field Documentation

13.1.2.1 color

```
HPDF_RGBColor color
```

Color of line

13.1.2.2 line_style

```
hpdftbl_line_style_t line_style
```

Line style (currently not used, preparation for future extensions)

13.1.2.3 width

```
HPDF_REAL width
```

Line width of border

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

13.2 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 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
- hpdftbl_content_callback_t label_cb
- hpdf_text_style_t content_style
- hpdftbl_content_callback_t content_cb
- hpdftbl_content_style_callback_t content_style_cb
- hpdftbl_canvas_callback_t canvas_cb
- hpdftbl_cell_t ∗ cells
- hpdf_border_style_t outer_border
- hpdf_border_style_t inner_border
- float ∗ col_width_percent

13.2.1 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

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, and tut_ex14.c.

13.2.2 Field Documentation

13.2.2.1 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

13.2.2.2 cells

```
hpdftbl_cell_t* cells
```

Reference to all an array of cells in the table

13.2.2.3 col_width_percent

```
float* col_width_percent
```

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

13.2.2.4 cols

```
size_t cols
```

Number of columns in table

Referenced by hpdftbl_set_row_content_style().

13.2.2.5 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

13.2.2.6 content_style

```
hpdf_text_style_t content_style
```

Content style

13.2.2.7 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

13.2.2.8 header_style

```
hpdf_text_style_t header_style
```

Header style

13.2.2.9 height

```
HPDF_REAL height
```

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

13.2.2.10 inner_border

```
hpdf_border_style_t inner_border
```

Table inner border settings

13.2.2.11 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

13.2.2.12 label_style

```
hpdf_text_style_t label_style
```

Label style settings

13.2.2.13 outer_border

```
hpdf_border_style_t outer_border
```

Table outer border settings

13.2.2.14 pdf_doc

```
HPDF_Doc pdf_doc
```

PDF document references

13.2.2.15 pdf_page

```
HPDF_Page pdf_page
```

PDF page reference

13.2.2.16 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()

13.2.2.17 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()

13.2.2.18 rows

```
size_t rows
```

Number of rows in table

Referenced by hpdftbl_set_col_content_style().

13.2.2.19 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

13.2.2.20 title_style

```
hpdf_text_style_t title_style
```

Title style

13.2.2.21 title_txt

```
char* title_txt
```

Title text

13.2.2.22 use_cell_labels

```
_Bool use_cell_labels
```

Flag to determine if cell labels should be used

13.2.2.23 use_header_row

```
_Bool use_header_row
```

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

13.2.2.24 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.

13.2.2.25 width

```
HPDF_REAL width
```

Table width

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

13.3 hpdftbl_cell Struct Reference

Specification of individual cells in the table.

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

Data Fields

- 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
- hpdftbl_content_callback_t label_cb
- hpdftbl_content_style_callback_t style_cb
- hpdftbl_canvas_callback_t canvas_cb
- hpdf_text_style_t content_style
- struct hpdftbl_cell ∗ parent_cell

13.3.1 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.

13.3.2 Field Documentation

13.3.2.1 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

13.3.2.2 colspan

```
size_t colspan
```

Number of column this cell spans

13.3.2.3 content

```
char* content
```

String reference for cell content

13.3.2.4 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

13.3.2.5 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

13.3.2.6 delta_x

```
HPDF_REAL delta_x
```

X-Position of cell from bottom left of table

13.3.2.7 delta_y

```
HPDF_REAL delta_y
```

Y-Position of cell from bottom left of table

13.3.2.8 height

```
HPDF_REAL height
```

Height of cell

13.3.2.9 label

```
char* label
```

String reference for label text

13.3.2.10 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

13.3.2.11 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.

13.3.2.12 rowspan

```
size_t rowspan
```

Number of rows this cell spans

13.3.2.13 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

13.3.2.14 textwidth

```
HPDF_REAL textwidth
```

Width of content string

13.3.2.15 width

```
HPDF_REAL width
```

Width of cells

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

13.4 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

13.4.1 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

tut_ex13_2.c.

13.4.2 Field Documentation

13.4.2.1 canvas_cb

```
hpdftbl_canvas_callback_t canvas_cb
```

Canvas callback for this cell

13.4.2.2 col

```
size_t col
```

Row for specified cell

13.4.2.3 colspan

```
unsigned colspan
```

Number of columns the specified cell should span

13.4.2.4 content_cb

```
hpdftbl_content_callback_t content_cb
```

Content callback for this cell

13.4.2.5 label

```
char* label
```

The label for this cell

13.4.2.6 label_cb

```
hpdftbl_content_callback_t label_cb
```

Label callback for this cell

13.4.2.7 row

```
size_t row
```

Row for specified cell

Examples

tut_ex13_2.c.

13.4.2.8 rowspan

```
unsigned rowspan
```

Number of rows the specified cell should span

13.4.2.9 style_cb

```
hpdftbl_content_style_callback_t style_cb
```

Content style callback for this cell

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

13.5 hpdftbl_errcode_entry Struct Reference

An entry in the error string table.

Data Fields

- char ∗ errstr
- unsigned errcode

13.5.1 Detailed Description

An entry in the error string table.

13.5.2 Field Documentation

13.5.2.1 errcode

```
unsigned errcode
```

The error code from HPDF library

13.5.2.2 errstr

```
char* errstr
```

Pointer to the error string

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl_errstr.c

13.6 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

13.6.1 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

tut_ex13_1.c, and tut_ex13_2.c.

13.6.2 Field Documentation

13.6.2.1 cell_spec

```
hpdftbl_cell_spec_t* cell_spec
```

Array of cell specification

13.6.2.2 cols

```
size_t cols
```

Number of columns in the table

Referenced by hpdftbl_stroke_from_data().

13.6.2.3 content_cb

```
hpdftbl_content_callback_t content_cb
```

Content callback for this table

13.6.2.4 height

```
HPDF_REAL height
```

Height of table

13.6.2.5 label_cb

```
hpdftbl_content_callback_t label_cb
```

Label callback for this table

13.6.2.6 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.

13.6.2.7 rows

```
size_t rows
```

Number of rows in the table

Referenced by hpdftbl_stroke_from_data().

13.6.2.8 style_cb

```
hpdftbl_content_style_callback_t style_cb
```

Content style callback for table

13.6.2.9 title

```
char* title
```

Table title

Examples

tut_ex13_1.c, and tut_ex13_2.c.

Referenced by hpdftbl_stroke_from_data().

13.6.2.10 use_header

```
_Bool use_header
```

Use a header for the table

13.6.2.11 use_labelgrid

```
_Bool use_labelgrid
```

Use label grid in table

13.6.2.12 use_labels

```
_Bool use_labels
```

Use labels in table

13.6.2.13 width

```
HPDF_REAL width
```

Width of table

13.6.2.14 xpos

```
HPDF_REAL xpos
```

X-position for table

13.6.2.15 ypos

```
HPDF_REAL ypos
```

Y-position for table

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

13.7 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
- hpdf_border_style_t * inner_border
- hpdf_border_style_t * outer_border
- _Bool use_labels
- _Bool use_label_grid_style
- _Bool use_header_row

13.7.1 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

13.7.2 Field Documentation

13.7.2.1 content_style

```
hpdf_text_style_t* content_style
```

Content text style

13.7.2.2 header_style

```
hpdf_text_style_t* header_style
```

Header text style

13.7.2.3 inner_border

```
hpdf_border_style_t* inner_border
```

Table inner border style

13.7.2.4 label_style

```
hpdf_text_style_t* label_style
```

Label text style

13.7.2.5 outer_border

```
hpdf_border_style_t* outer_border
```

Table outer border style

13.7.2.6 title_style

```
hpdf_text_style_t* title_style
```

Table title text style

13.7.2.7 use_header_row

```
_Bool use_header_row
```

Flag if header row should be used

13.7.2.8 use_label_grid_style

```
_Bool use_label_grid_style
```

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

13.7.2.9 use_labels

```
_Bool use_labels
```

Flag if cell labels should be used

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

13.8 line_dash_style Struct Reference

Definition of a dashed line style.

Data Fields

- HPDF_UINT16 dash_ptn [8]
- size_t num

13.8.1 Detailed Description

Definition of a dashed line style.

13.8.2 Field Documentation

13.8.2.1 dash_ptn

```
HPDF_UINT16 dash_ptn[8]
```

HPDF ash line definition

13.8.2.2 num

```
size_t num
```

Number of segments in the dashed line

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.c

13.9 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

13.9.1 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.

13.9.2 Field Documentation

13.9.2.1 background

```
HPDF_RGBColor background
```

Font background color

Examples

tut_ex09.c.

13.9.2.2 color

```
HPDF_RGBColor color
```

Font color

Examples

tut_ex09.c.

13.9.2.3 font

```
char* font
```

Font face name

Examples

tut_ex09.c.

13.9.2.4 fsize

```
HPDF_REAL fsize
```

Font size

Examples

tut_ex09.c.

13.9.2.5 halign

```
hpdftbl_text_align_t halign
```

Text horizontal alignment

Examples

tut_ex09.c.

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

- /Users/ljp/Devel/hpdf_table/src/hpdftbl.h

Chapter 14

File Documentation

14.1 config.h

```
/* src/config.h.  Generated from config.h.in by configure.  */
/* src/config.h.in.  Generated from configure.ac by autoheader.  */

/* Define to 1 if you have the <dlfcn.h> header file. */
#define HAVE_DLFCN_H 1

/* Define to 1 if you have the <hpdf.h> header file. */
#define HAVE_HPDF_H 1

/* Define to 1 if you have the <iconv.h> header file. */
#define HAVE_ICONV_H 1

/* Define to 1 if you have the <inttypes.h> header file. */
#define HAVE_INTTYPES_H 1

/* Define to 1 if you have the ‘hpdf’ library (-lhpdf). */
#define HAVE_LIBHPDF 1

/* Define to 1 if you have the ‘iconv’ library (-liconv). */
#define HAVE_LIBICONV 1

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdio.h> header file. */
#define HAVE_STDIO_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1

/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the <sys/stat.h> header file. */
#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/types.h> header file. */
#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have the <unistd.h> header file. */
#define HAVE_UNISTD_H 1

/* True if system type is Apple OSX */
#define IS_OSX 1

/* Define to the sub-directory where libtool stores uninstalled libraries. */
#define LT_OBJDIR ".libs/"

/* Name of package */
#define PACKAGE "libhpdftbl"

/* Define to the address where bug reports for this package should be sent. */
#define PACKAGE_BUGREPORT "johan162@gmail.com"

/* Define to the full name of this package. */
```

```
#define PACKAGE_NAME "libhpdftbl"

/* Define to the full name and version of this package. */
#define PACKAGE_STRING "libhpdftbl 1.0.0-beta2"

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME "libhpdftbl"

/* Define to the home page for this package. */
#define PACKAGE_URL ""

/* Define to the version of this package. */
#define PACKAGE_VERSION "1.0.0-beta2"

/* Define to 1 if all of the C90 standard headers exist (not just the ones
   required in a freestanding environment). This macro is provided for
   backward compatibility; new code need not use it. */
#define STDC_HEADERS 1

/* Version number of package */
#define VERSION "1.0.0-beta2"
```

14.2 /Users/ljp/Devel/hpdf_table/src/hpdftbl.c File Reference

Main source module for hpdftbl.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <iconv.h>
#include <hpdf.h>
#include "hpdftbl.h"
```

Data Structures

- struct line_dash_style

    Definition of a dashed line style.

Macros

- #define **TRUE** 1
- #define **FALSE** 0
- #define **_IDX**(r, c) (r∗t->cols+c)
- #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}
- #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}
- #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}
- #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}
- #define **HPDFTBL_DEFAULT_INNER_BORDER_STYLE** (hpdf_border_style_t){0.7f, (HPDF_RGBColor){0.←5f,0.5f,0.5f},0}
- #define **HPDFTBL_DEFAULT_OUTER_BORDER_STYLE** (hpdf_border_style_t){1.0f, (HPDF_RGBColor){0.←2f,0.2f,0.2f},0}
- #define **_SET_ERR**(t, err, r, c) do {err_code=err;err_row=r;err_col=c; if(hpdftbl_err_handler){hpdftbl_err_←handler(t,r,c,err);}} while(0)
- #define **_CHK_TABLE**(t) do {if(NULL == t) {err_code=-3;err_row=-1;err_col=-1;return -1;}} while(0)
- #define **ERR_UNKNOWN** 11

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_style_t style)

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

    Switch stroking anchor point.
- _Bool hpdftbl_get_anchor_top_left (void)

    Get stroking anchor point.
- 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 simple default table error handler callback that outputs the error to stderr in human readable format and quits the process.
- int hpdftbl_get_last_errcode (const char ∗∗errstr, int ∗row, int ∗col)

    Return last error code.
- hpdftbl_error_handler_t hpdftbl_set_errhandler (hpdftbl_error_handler_t err_handler)

    Specify errhandler for the table routines.
- 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.
- 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.
- 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_colwidth_percent (hpdftbl_t t, size_t c, float w)

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

    Specify style for table outer border.
- int hpdftbl_set_inner_border (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color)

    Specify style for table inner border.
- int hpdftbl_set_header_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_←
RGBColor background)

    Specify style for table heder 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 text align.
- int hpdftbl_use_header (hpdftbl_t t, _Bool use)

- int hpdftbl_use_labels (hpdftbl_t t, _Bool use)
- int hpdftbl_use_labelgrid (hpdftbl_t t, _Bool use)
- 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.
- int hpdftbl_set_cell (hpdftbl_t t, int r, int 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_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_canvas_cb (hpdftbl_t t, hpdftbl_canvas_callback_t cb)

  Set cell canvas callback.
- 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 font style for labels.
- int hpdftbl_set_content_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set font 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 font 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 font style for an entre 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 font style for content of specified cell.
- 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.
- int hpdftbl_set_title_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set the table title 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_apply_theme (hpdftbl_t t, hpdftbl_theme_t ∗theme)

  Apply a specified theme to a 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_get_last_auto_height (HPDF_REAL ∗height)

  Get the height calculated for the last constructed 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.

14.2.1 Detailed Description

Main source module for hpdftbl.

14.2.2 Function Documentation

14.2.2.1 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().

14.2.2.2 hpdftbl_apply_theme()

```
int hpdftbl_apply_theme (
            hpdftbl_t t,
            hpdftbl_theme_t * theme )
```

Apply a specified theme to a table.

Apply a specified theme to a table. The default table 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()

14.2.2.3 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()

14.2.2.4 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, and tut_ex12.c.

14.2.2.5 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

tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex14.c.

Referenced by hpdftbl_create(), and hpdftbl_stroke_from_data().

14.2.2.6 hpdftbl_default_table_error_handler()

```
void hpdftbl_default_table_error_handler (
            hpdftbl_t t,
            int r,
            int c,
            int err )
```

A simple default table error handler 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

Examples

tut_ex10.c, tut_ex11.c, and tut_ex12.c.

14.2.2.7 hpdftbl_destroy()

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

Destroy a table and free all memory.

Destroy a table previous created with table_create()

Parameters

t	Handle to table

Returns

0 on success, -1 on failure

14.2.2.8 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

14.2.2.9 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

14.2.2.10 hpdftbl_get_anchor_top_left()

```
_Bool hpdftbl_get_anchor_top_left (
             void )
```

Get stroking anchor point.

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

See also

hpdftbl_set_anchor_top_left

14.2.2.11 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

See also

hpdftbl_apply_theme()

14.2.2.12 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_code	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().

14.2.2.13 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 heigh 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

14.2.2.14 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

14.2.2.15 hpdftbl_set_anchor_top_left()

```
void hpdftbl_set_anchor_top_left (
             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 set the basepoint to bottom left instead.

Parameters

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

14.2.2.16 hpdftbl_set_background()

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

Set table background color.

Set table background

Parameters

t	Table handle
background	Background color

Returns

0 on success, -1 on failure

14.2.2.17 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 sepcific cell use the hpdftbl_set_cell_canvas_callback() function

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_set_cell_canvas_cb()

14.2.2.18 hpdftbl_set_cell()

```
int hpdftbl_set_cell (
            hpdftbl_t t,
            int r,
            int 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 then error is given (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.

14.2.2.19 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_callback

Examples

tut_ex14.c.

14.2.2.20 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.

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.

14.2.2.21 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 font 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().

14.2.2.22 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()

14.2.2.23 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 content callback.

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()

14.2.2.24 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

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

tut_ex07.c, and tut_ex08.c.

14.2.2.25 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 font style for an entre 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()

14.2.2.26 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.

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

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

14.2.2.27 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

tut_ex02.c, tut_ex02_1.c, tut_ex04.c, tut_ex05.c, tut_ex10.c, tut_ex11.c, and tut_ex12.c.

14.2.2.28 hpdftbl_set_content_cb()

```
int hpdftbl_set_content_cb (
            hpdftbl_t t,
            hpdftbl_content_callback_t cb )
```

Set table content callback.

Set 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

Parameters

t	Table handle
cb	Callback function

See also

hpdftbl_set_cell_content_cb()

Examples

tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex09.c.

14.2.2.29 hpdftbl_set_content_style()

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

Set font style for text content.

Set font options for cell content. This will be applied for all cells in the table.

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()

14.2.2.30 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.

14.2.2.31 hpdftbl_set_errhandler()

```
hpdftbl_error_handler_t hpdftbl_set_errhandler (
            hpdftbl_error_handler_t err_handler )
```

Specify errhandler for the table routines.

Parameters

err_handler	

Returns

The old error handler or NULL if non exists

Examples

tut_ex10.c, tut_ex11.c, and tut_ex12.c.

14.2.2.32 hpdftbl_set_header_halign()

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

Set table header text align.

Set horizontal text alignment for header row

Parameters

t	Table handle
align	Alignment

Returns

0 on success, -1 on failure

14.2.2.33 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 heder 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 hpdftbl_use_header()

14.2.2.34 hpdftbl_set_inner_border()

```
int hpdftbl_set_inner_border (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color )
```

Specify style for table inner border.

Set inner border properties

Parameters

t	Table handle
width	Line width
color	Line color

Returns

0 on success, -1 on failure

14.2.2.35 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.

14.2.2.36 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 font style for labels.

Set font, color and background options for cell labels.

Parameters

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

Returns

-1 on error, 0 if successful

14.2.2.37 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

tut_ex04.c, and tut_ex05.c.

14.2.2.38 hpdftbl_set_line_dash()

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

Internal helper to set the line style.

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

Parameters

t	Table handle
style	

Returns

-1 on error, 0 on success

See also

line_dash_style

14.2.2.39 hpdftbl_set_outer_border()

```
int hpdftbl_set_outer_border (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color )
```

Specify style for table outer border.

Set outer border properties

Parameters

t	Table handle
width	Line width
color	Line color

Returns

0 on success, -1 on failure

14.2.2.40 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 font 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()

14.2.2.41 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

14.2.2.42 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 charactes 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.

14.2.2.43 hpdftbl_set_title()

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

Set table title.

Set table title

Parameters

t	Table handle
title	Title string

Returns

0 on success, -1 on failure

See also

hpdftbl_set_title_style()

hpdftbl_set_title_halign()

14.2.2.44 hpdftbl_set_title_halign()

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

Set horizontal alignment for table title.

Set horizontal text alignment for title

Parameters

t	Table handle
align	Alignment

Returns

0 on success, -1 on failure

See also

hpdftbl_set_title()

hpdftbl_set_title_style()

14.2.2.45 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 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()

14.2.2.46 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, bottom left corner
ypos	y position for table, bottom left corner
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

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, and tut_ex14.c.

14.2.2.47 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

tut_ex13_1.c, and tut_ex13_2.c.

14.2.2.48 hpdftbl_use_header()

```
int hpdftbl_use_header (
             hpdftbl_t t,
             _Bool use )
```

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

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

tut_ex02_1.c, tut_ex11.c, and tut_ex12.c.

14.2.2.49 hpdftbl_use_labelgrid()

```
int hpdftbl_use_labelgrid (
            hpdftbl_t t,
            _Bool use )
```

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

tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex14.c.

14.2.2.50 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

tut_ex03.c, tut_ex04.c, tut_ex05.c, tut_ex06.c, tut_ex07.c, tut_ex08.c, and tut_ex14.c.

14.3 /Users/ljp/Devel/hpdf_table/src/hpdftbl.h File Reference

Necessary header file for HPDF table usage.

Data Structures

- struct text_style

  Specification of a text style.
- struct border_style

  Specification for table borders.
- 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 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 COLOR_DARK_RED (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
- #define **COLOR_RED** (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
- #define **COLOR_LIGHT_GREEN** (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
- #define **COLOR_GREEN** (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
- #define **COLOR_DARK_GREEN** (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
- #define **COLOR_DARK_GRAY** (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
- #define **COLOR_LIGHT_GRAY** (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
- #define **COLOR_GRAY** (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
- #define **COLOR_SILVER** (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
- #define **COLOR_LIGHT_BLUE** (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
- #define **COLOR_BLUE** (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
- #define **COLOR_DARK_BLUE** (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
- #define **COLOR_WHITE** (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
- #define **COLOR_BLACK** (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
- #define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"
- #define **HPDFTBL_DEFAULT_SOURCE_ENCODING** "UTF-8"
- #define **HPDFTBL_TEXT_HALIGN_LEFT** 0
- #define **HPDFTBL_TEXT_HALIGN_CENTER** 1
- #define **HPDFTBL_TEXT_HALIGN_RIGHT** 2
- #define A4PAGE_HEIGHT_CM 29.7
- #define A4PAGE_WIDTH_CM 21.0
- #define A3PAGE_HEIGHT_CM 42.0
- #define A3PAGE_WIDTH_CM 29.7
- #define LETTERRPAGE_HEIGHT_CM 27.9
- #define LETTERRPAGE_WIDTH_CM 21.6
- #define LEGALPAGE_HEIGHT_CM 35.6
- #define LEGALPAGE_WIDTH_CM 21.6
- #define HPDFTBL_END_CELLSPECS {0, 0, 0, 0, 0, 0, 0, 0, 0}
- #define HPDF_COLOR_FROMRGB(r, g, b) (HPDF_RGBColor){(r)/255.0,(g)/255.0,(b)/255.0}
- #define MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0
- #define hpdftbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54∗72)

    Convert cm to dots using the default resolution (72 DPI)

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 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 enum hpdftbl_dash_style **hpdftbl_line_style_t**

  Possible line dash styles in table frames.
- typedef struct border_style hpdf_border_style_t

  Specification for table borders.
- typedef struct hpdftbl_cell hpdftbl_cell_t

  Type definition for the cell structure.
- typedef struct hpdftbl ∗ hpdftbl_t

  Table handle is a pointer to the hpdftbl structure.
- typedef void(∗ hpdftbl_callback_t) (hpdftbl_t)

  Callback type for optional post processing when constructing table from a data array.
- 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_dash_style {
  SOLID = 0 , DOT1 = 1 , DOT2 = 2 , DOT3 = 3 ,
  DASH1 = 4 , DASH2 = 5 , DASH3 = 6 , DASHDOT = 7 }

  Possible line dash styles in table frames.

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_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_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 (_Bool anchor)

  Switch stroking anchor point.
- _Bool hpdftbl_get_anchor_top_left (void)

  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_default_table_error_handler (hpdftbl_t t, int r, int c, int err)

  A simple default table error handler callback that outputs the error to stderr in human readable format and quits the process.
- 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_destroy_theme (hpdftbl_theme_t ∗theme)

  Destroy existing theme structure and free memory.
- 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_use_labels (hpdftbl_t t, _Bool use)
- int hpdftbl_use_labelgrid (hpdftbl_t t, _Bool use)
- int hpdftbl_set_background (hpdftbl_t t, HPDF_RGBColor background)

  Set table background color.
- int hpdftbl_set_outer_border (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color)

  Specify style for table outer border.
- int hpdftbl_set_inner_border (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color)

  Specify style for table inner border.
- int hpdftbl_set_header_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Specify style for table heder row.
- int hpdftbl_set_header_halign (hpdftbl_t t, hpdftbl_text_align_t align)

  Set table header text align.
- int hpdftbl_use_header (hpdftbl_t t, _Bool use)
- int hpdftbl_set_label_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set the font style for labels.

- 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 font 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 font style for an entre column of cells.
- int hpdftbl_set_content_style (hpdftbl_t t, char ∗font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↩ RGBColor background)

  Set font 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 font 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 style.
- int hpdftbl_set_cell (hpdftbl_t t, int r, int 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_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.
- 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.
- 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.

14.3.1 Detailed Description

Necessary header file for HPDF table usage.

14.3.2 Macro Definition Documentation

14.3.2.1 A3PAGE_HEIGHT_CM

```
#define A3PAGE_HEIGHT_CM 42.0
```

A3 Height in CM

14.3.2.2 A3PAGE_WIDTH_CM

```
#define A3PAGE_WIDTH_CM 29.7
```

A3 Width in CM

14.3.2.3 A4PAGE_HEIGHT_CM

```
#define A4PAGE_HEIGHT_CM 29.7
```

A4 Height in CM

Examples

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, and tut_ex14.c.

14.3.2.4 A4PAGE_WIDTH_CM

```
#define A4PAGE_WIDTH_CM 21.0
```

A4 Width in CM

Examples

tut_ex02_1.c, tut_ex09.c, tut_ex10.c, tut_ex11.c, and tut_ex12.c.

14.3.2.5 COLOR_DARK_RED

```
#define COLOR_DARK_RED (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
```

Basic color definitions

Examples

tut_ex14.c.

14.3.2.6 HPDF_COLOR_FROMRGB

```
#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]

14.3.2.7 HPDF_FF_TIMES

```
#define HPDF_FF_TIMES "Times-Roman"
```

Definition of built-in HPDF font families

Examples

tut_ex09.c.

14.3.2.8 hpdftbl_cm2dpi

```
#define hpdftbl_cm2dpi(
              c ) (((HPDF_REAL)(c))/2.54*72)
```

Convert cm to dots using the default resolution (72 DPI)

Parameters

cm	Measure in cm

Returns

HPDF_REAL Converted value in dots

Examples

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, and tut_ex14.c.

14.3.2.9 HPDFTBL_DEFAULT_TARGET_ENCODING

```
#define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"
```

Text encodings

14.3.2.10 HPDFTBL_END_CELLSPECS

```
#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

Examples

tut_ex13_2.c.

14.3.2.11 LEGALPAGE_HEIGHT_CM

```
#define LEGALPAGE_HEIGHT_CM 35.6
```

US Legal Height in CM

14.3.2.12 LEGALPAGE_WIDTH_CM

```
#define LEGALPAGE_WIDTH_CM 21.6
```

US Legal Width in CM

14.3.2.13 LETTERRPAGE_HEIGHT_CM

```
#define LETTERRPAGE_HEIGHT_CM 27.9
```

US Letter Height in CM

14.3.2.14 LETTERRPAGE_WIDTH_CM

```
#define LETTERRPAGE_WIDTH_CM 21.6
```

US Letter Width in CM

14.3.2.15 MIN_CALCULATED_PERCENT_CELL_WIDTH

```
#define MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0
```

The smallest size in percent of table width allowed by automatic calculation before giving an error

14.3.3 Typedef Documentation

14.3.3.1 hpdf_border_style_t

```
typedef struct border_style hpdf_border_style_t
```

Specification for table borders.

Contains line properties used when stroking a border line

14.3.3.2 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)

14.3.3.3 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()

14.3.3.4 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_callback()

14.3.3.5 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()

14.3.3.6 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.

14.3.3.7 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_callback()

14.3.3.8 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_callback()

14.3.3.9 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()

14.3.3.10 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.

14.3.3.11 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()

14.3.3.12 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

14.3.3.13 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

14.3.4 Enumeration Type Documentation

14.3.4.1 hpdftbl_dash_style

```
enum hpdftbl_dash_style
```

Possible line dash styles in table frames.

Enumerator

SOLID	Solid line
DOT1	Dotted line variant 1
DOT2	Dotted line variant 2
DOT3	Dotted line variant 3
DASH1	Dashed line variant 1
DASH2	Dashed line variant 2
DASH3	Dashed line variant 3
DASHDOT	Dashed-dot line variant 1

14.3.4.2 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

14.3.5 Function Documentation

14.3.5.1 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().

14.3.5.2 hpdftbl_apply_theme()

```
int hpdftbl_apply_theme (
             hpdftbl_t t,
             hpdftbl_theme_t * theme )
```

Apply a specified theme to a table.

Apply a specified theme to a table. The default table 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()

14.3.5.3 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()

14.3.5.4 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

14.3.5.5 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

Referenced by hpdftbl_create(), and hpdftbl_stroke_from_data().

14.3.5.6 hpdftbl_default_table_error_handler()

```
void hpdftbl_default_table_error_handler (
            hpdftbl_t t,
            int r,
            int c,
            int err )
```

A simple default table error handler 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

14.3.5.7 hpdftbl_destroy()

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

Destroy a table and free all memory.

Destroy a table previous created with table_create()

Parameters

t	Handle to table

Returns

0 on success, -1 on failure

14.3.5.8 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

14.3.5.9 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

14.3.5.10 hpdftbl_get_anchor_top_left()

```
_Bool hpdftbl_get_anchor_top_left (
            void )
```

Get stroking anchor point.

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

See also

hpdftbl_set_anchor_top_left

14.3.5.11 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

See also

hpdftbl_apply_theme()

14.3.5.12 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_code	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().

14.3.5.13 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 heigh 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

14.3.5.14 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

14.3.5.15 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()

Examples

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, and tut_ex14.c.

14.3.5.16 hpdftbl_set_anchor_top_left()

```
void hpdftbl_set_anchor_top_left (
             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 set the basepoint to bottom left instead.

Parameters

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

14.3.5.17 hpdftbl_set_background()

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

Set table background color.

Set table background

Parameters

t	Table handle
background	Background color

Returns

0 on success, -1 on failure

14.3.5.18 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 sepcific cell use the hpdftbl_set_cell_canvas_callback() function

Parameters

t	Table handle
cb	Callback function

Returns

-1 on failure, 0 otherwise

See also

hpdftbl_set_cell_canvas_cb()

14.3.5.19 hpdftbl_set_cell()

```
int hpdftbl_set_cell (
            hpdftbl_t t,
            int r,
            int 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 then error is given (returns -1),

Parameters

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

Returns

-1 on error, 0 if successful

14.3.5.20 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_callback

14.3.5.21 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.

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()

14.3.5.22 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 font 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().

14.3.5.23 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()

14.3.5.24 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 content callback.

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()

14.3.5.25 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

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()

14.3.5.26 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 font style for an entre 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()

14.3.5.27 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.

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

14.3.5.28 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()

14.3.5.29 hpdftbl_set_content_cb()

```
int hpdftbl_set_content_cb (
            hpdftbl_t t,
            hpdftbl_content_callback_t cb )
```

Set table content callback.

Set 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

Parameters

t	Table handle
cb	Callback function

See also

hpdftbl_set_cell_content_cb()

14.3.5.30 hpdftbl_set_content_style()

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

Set font style for text content.

Set font options for cell content. This will be applied for all cells in the table.

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()

14.3.5.31 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()

14.3.5.32 hpdftbl_set_errhandler()

```
hpdftbl_error_handler_t hpdftbl_set_errhandler (
            hpdftbl_error_handler_t err_handler )
```

Specify errhandler for the table routines.

Parameters

err_handler	

Returns

The old error handler or NULL if non exists

14.3.5.33 hpdftbl_set_header_halign()

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

Set table header text align.

Set horizontal text alignment for header row

Parameters

t	Table handle
align	Alignment

Returns

0 on success, -1 on failure

14.3.5.34 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 heder 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 hpdftbl_use_header()

14.3.5.35 hpdftbl_set_inner_border()

```
int hpdftbl_set_inner_border (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color )
```

Specify style for table inner border.

Set inner border properties

Parameters

t	Table handle
width	Line width
color	Line color

Returns

0 on success, -1 on failure

14.3.5.36 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()

14.3.5.37 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 font style for labels.

Set font, color and background options for cell labels.

Parameters

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

Returns

-1 on error, 0 if successful

14.3.5.38 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()

14.3.5.39 hpdftbl_set_outer_border()

```
int hpdftbl_set_outer_border (
            hpdftbl_t t,
            HPDF_REAL width,
            HPDF_RGBColor color )
```

Specify style for table outer border.

Set outer border properties

Parameters

t	Table handle
width	Line width
color	Line color

Returns

0 on success, -1 on failure

14.3.5.40 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 font 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

Parameters

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()

14.3.5.41 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

14.3.5.42 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 charactes 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.

14.3.5.43 hpdftbl_set_title()

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

Set table title.

Set table title

Parameters

t	Table handle
title	Title string

Returns

0 on success, -1 on failure

See also

hpdftbl_set_title_style()

hpdftbl_set_title_halign()

14.3.5.44 hpdftbl_set_title_halign()

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

Set horizontal alignment for table title.

Set horizontal text alignment for title

Parameters

t	Table handle
align	Alignment

Returns

0 on success, -1 on failure

See also

hpdftbl_set_title()

hpdftbl_set_title_style()

14.3.5.45 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 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()

14.3.5.46 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, bottom left corner
ypos	y position for table, bottom left corner
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()

14.3.5.47 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()

14.3.5.48 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

Examples

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, and tut_ex14.c.

14.3.5.49 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)

14.3.5.50 hpdftbl_use_header()

```
int hpdftbl_use_header (
              hpdftbl_t t,
              _Bool use )
```

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

Parameters

t	Table handle
use	TRUE to enable, FALSE to disable

Returns

0 on success, -1 on failure

See also

hpdftbl_set_header_style()

14.3.5.51 hpdftbl_use_labelgrid()

```
int hpdftbl_use_labelgrid (
            hpdftbl_t t,
            _Bool use )
```

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

14.3.5.52 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()

14.3.5.53 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

14.3.5.54 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

tut_ex14.c.

14.3.5.55 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

14.3.5.56 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

tut_ex14.c.

14.4 hpdftbl.h

Go to the documentation of this file.

```
/* =========================================================================
 * File:        hpdftbl.h
 * Description: Utility module for flexible table drawing with HPDF library
 * Author:      Johan Persson (johan162@gmail.com)
 *
 * Copyright (C) 2022 Johan Persson
 *
 * 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, INCLUDING 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.
 * ==========================================================================
 */

#ifndef hpdftbl_H
#define    hpdftbl_H

#ifdef    __cplusplus
// in case we have C++ code, we should use its' types and logic
#include <algorithm>
typedef std::_Bool _Bool;
#endif

#ifdef    __cplusplus
extern "C" {
#endif

#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 COLOR_DARK_RED      (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
#define COLOR_RED           (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
#define COLOR_LIGHT_GREEN   (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
#define COLOR_GREEN         (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
#define COLOR_DARK_GREEN    (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
#define COLOR_DARK_GRAY     (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
#define COLOR_LIGHT_GRAY    (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
#define COLOR_GRAY          (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
#define COLOR_SILVER        (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
#define COLOR_LIGHT_BLUE    (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
#define COLOR_BLUE          (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
#define COLOR_DARK_BLUE     (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
#define COLOR_WHITE         (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
#define COLOR_BLACK         (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }

#define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"
#define HPDFTBL_DEFAULT_SOURCE_ENCODING "UTF-8"

#define HPDFTBL_TEXT_HALIGN_LEFT 0
#define HPDFTBL_TEXT_HALIGN_CENTER 1
#define HPDFTBL_TEXT_HALIGN_RIGHT 2

/*
 * Standard paper heights
 */
#define A4PAGE_HEIGHT_CM 29.7
#define A4PAGE_WIDTH_CM 21.0
#define A3PAGE_HEIGHT_CM 42.0
#define A3PAGE_WIDTH_CM 29.7
#define LETTERRPAGE_HEIGHT_CM 27.9
#define LETTERRPAGE_WIDTH_CM 21.6
#define LEGALPAGE_HEIGHT_CM 35.6
#define LEGALPAGE_WIDTH_CM 21.6
#define HPDFTBL_END_CELLSPECS {0, 0, 0, 0, 0, 0, 0, 0, 0}

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

#define MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0

#define hpdftbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54*72)

typedef enum hpdftbl_text_align {
    LEFT = 0,
    CENTER = 1,
    RIGHT = 2
} hpdftbl_text_align_t;

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

```
} hpdf_text_style_t;

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

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

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

typedef enum hpdftbl_dash_style {
    SOLID = 0,
    DOT1 = 1,
    DOT2 = 2,
    DOT3 = 3,
    DASH1 = 4,
    DASH2 = 5,
    DASH3 = 6,
    DASHDOT = 7
} hpdftbl_line_style_t;

typedef struct border_style {
    HPDF_REAL width;
    HPDF_RGBColor color;
    hpdftbl_line_style_t line_style;
} hpdf_border_style_t;

struct hpdftbl_cell {
    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;
    hpdftbl_content_callback_t label_cb;
    hpdftbl_content_style_callback_t style_cb;
    hpdftbl_canvas_callback_t canvas_cb;
    hpdf_text_style_t content_style;
    struct hpdftbl_cell *parent_cell;
};

typedef struct hpdftbl_cell hpdftbl_cell_t;

struct hpdftbl {
    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 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;
    hpdftbl_content_callback_t label_cb;
    hpdf_text_style_t content_style;
    hpdftbl_content_callback_t content_cb;
    hpdftbl_content_style_callback_t content_style_cb;
    hpdftbl_canvas_callback_t canvas_cb;
    hpdftbl_cell_t *cells;
    hpdf_border_style_t outer_border;
    hpdf_border_style_t inner_border;
    float *col_width_percent;
};

typedef struct hpdftbl *hpdftbl_t;

typedef void (*hpdftbl_callback_t)(hpdftbl_t);

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;

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;

typedef struct hpdftbl_theme {
    hpdf_text_style_t *content_style;
    hpdf_text_style_t *label_style;
    hpdf_text_style_t *header_style;
    hpdf_text_style_t *title_style;
    hpdf_border_style_t *inner_border;
    hpdf_border_style_t *outer_border;
    _Bool use_labels;
    _Bool use_label_grid_style;
    _Bool use_header_row;
} hpdftbl_theme_t;

typedef void (*hpdftbl_error_handler_t)(hpdftbl_t, int, int, int);

static hpdftbl_error_handler_t hpdftbl_err_handler = NULL;

/*
 * Table creation and destruction function
 */
hpdftbl_t
hpdftbl_create(size_t rows, size_t cols);

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

int
hpdftbl_stroke(HPDF_Doc pdf,
               HPDF_Page page, hpdftbl_t t,
               HPDF_REAL xpos, HPDF_REAL ypos,
               HPDF_REAL width, HPDF_REAL height);

int
hpdftbl_stroke_from_data(HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t *tbl_spec, hpdftbl_theme_t
    *theme);

int
hpdftbl_destroy(hpdftbl_t t);

int
hpdftbl_get_last_auto_height(HPDF_REAL *height);

void
hpdftbl_set_anchor_top_left(_Bool anchor);

_Bool
hpdftbl_get_anchor_top_left(void);

/*
 * Table error handling functions
 */
hpdftbl_error_handler_t
hpdftbl_set_errhandler(hpdftbl_error_handler_t);

const char *
hpdftbl_get_errstr(int err);

const char *
hpdftbl_hpdf_get_errstr(HPDF_STATUS err_code);

int
hpdftbl_get_last_errcode(const char **errstr, int *row, int *col);

void
hpdftbl_default_table_error_handler(hpdftbl_t t, int r, int c, int err);
```

```

/*
 * Theme handling functions
 */
int
hpdftbl_apply_theme(hpdftbl_t t, hpdftbl_theme_t *theme);

hpdftbl_theme_t *
hpdftbl_get_default_theme(void);

int
hpdftbl_destroy_theme(hpdftbl_theme_t *theme);

/*
 * Table layout adjusting functions
 */
int
hpdftbl_set_colwidth_percent(hpdftbl_t t, size_t c, float w);

int
hpdftbl_clear_spanning(hpdftbl_t t);

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

/*
 * Table style handling functions
 */
int
hpdftbl_use_labels(hpdftbl_t t, _Bool use);

int
hpdftbl_use_labelgrid(hpdftbl_t t, _Bool use);

int
hpdftbl_set_background(hpdftbl_t t, HPDF_RGBColor background);

int
hpdftbl_set_outer_border(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color);

int
hpdftbl_set_inner_border(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color);

int
hpdftbl_set_header_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
    background);

int
hpdftbl_set_header_halign(hpdftbl_t t, hpdftbl_text_align_t align);

int
hpdftbl_use_header(hpdftbl_t t, _Bool use);

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

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

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

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

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);

int
hpdftbl_set_title_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
    background);

/*
 * Table content handling
 */
int
hpdftbl_set_cell(hpdftbl_t t, int r, int c, char *label, char *content);

int
hpdftbl_set_tag(hpdftbl_t t, void *tag);

```

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

int
hpdftbl_set_title_halign(hpdftbl_t t, hpdftbl_text_align_t align);

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

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

/*
 * Table callback functions
 */
int
hpdftbl_set_content_cb(hpdftbl_t t, hpdftbl_content_callback_t cb);

int
hpdftbl_set_cell_content_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb);

int
hpdftbl_set_cell_content_style_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_style_callback_t cb);

int
hpdftbl_set_content_style_cb(hpdftbl_t t, hpdftbl_content_style_callback_t cb);

int
hpdftbl_set_label_cb(hpdftbl_t t, hpdftbl_content_callback_t cb);

int
hpdftbl_set_cell_label_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb);

int
hpdftbl_set_canvas_cb(hpdftbl_t t, hpdftbl_canvas_callback_t cb);

int
hpdftbl_set_cell_canvas_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_canvas_callback_t cb);

/*
 * Text encoding
 */
void
hpdftbl_set_text_encoding(char *target, char *source);

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

/*
 * Misc utility and widget functions
 */

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

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);

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);

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);

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);

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);

#ifdef    __cplusplus
}
#endif

#endif    /* hpdftbl_H */
```

14.5 /Users/ljp/Devel/hpdf_table/src/hpdftbl_errstr.c File Reference

Utility module to translate HPDF error codes to human readable strings.

```
#include <hpdf.h>
```

Data Structures

- struct hpdftbl_errcode_entry

  An entry in the error string table.

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.

14.5.1 Detailed Description

Utility module to translate HPDF error codes to human readable strings.

14.5.2 Function Documentation

14.5.2.1 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()

14.6 /Users/ljp/Devel/hpdf_table/src/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)

14.6.1 Detailed Description

Create a grid on a document for positioning.

14.6.2 Function Documentation

14.6.2.1 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

14.7 /Users/ljp/Devel/hpdf_table/src/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.

14.7.1 Detailed Description

Support for drawing widgets.

14.7.2 Function Documentation

14.7.2.1 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
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)

14.7.2.2 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

14.7.2.3 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

14.7.2.4 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

14.7.2.5 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

Parameters

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

Chapter 15

Example Documentation

15.1 tut_ex01.c

The very most basic table with API call to set content in each cell.

```
#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>
#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 "tut_ex01.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex01.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex01(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, 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);
}
// Setup a new PDF document with one age
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex01(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.2 tut_ex02.c

Basic table with content data specified as an array.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex02.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex02.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#ifndef _MSC_VER
```

```c
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(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++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
void
create_table_ex02(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;
    setup_dummy_data(&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
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex02(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
```

```
#endif
```

15.3 tut_ex02_1.c

Basic table with content data specified as an array.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#endif
#include <syslog.h>
// 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 "tut_ex02_1.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex02_1.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(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_data(&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);
}
// Setup a new PDF document with one page
```

```
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        closelog();
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex02_1(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.4 tut_ex03.c

First example with API call to set content in each cell with added labels and shortened grid.

```
#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>
#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 "tut_ex03.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex03.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
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);
}
// Setup a new PDF document with one age
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex03(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.5 tut_ex04.c

Specifying labels as data array

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
```

```c
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex04.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex04.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(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++;
        }
    }
}
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_data(&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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex04(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.6 tut_ex05.c

Set content data specified as an array with added labels and shortened grid.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex05.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex05.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(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++;
        }
    }
}
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_data(&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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex05(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.7 tut_ex06.c

Use content to set content and labels.

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

```
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex06.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex06.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
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;
}
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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex06(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.8 tut_ex07.c

Expand cells over multiple columns and rows.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex07.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex07.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
```

```
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
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;
}
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);
    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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex07(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.9 tut_ex08.c

Adjust column width and expand cells over multiple columns and rows.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex08.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex08.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
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;
}
```

```
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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex08(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.10 tut_ex09.c

Adjusting font style with a callback.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex09.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex09.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
_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 = COLOR_BLACK;
        style->background = 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 = COLOR_BLACK;
        style->background = 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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex09(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.11 tut_ex10.c

Adjust column widths and add error handler.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#endif
#include <syslog.h>
// 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 "tut_ex10.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex10.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(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++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
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, 40);
    hpdftbl_set_colwidth_percent(tbl, 1, 70);
    content_t content;
    setup_dummy_data(&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
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        closelog();
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex10(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.12 tut_ex11.c

Table with header row and error handler.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#endif
#include <syslog.h>
// 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 "tut_ex11.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex11.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
```

```
#endif
typedef char **content_t;
void setup_dummy_data(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++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
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_data(&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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        closelog();
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex11(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.13 tut_ex12.c

Table with header row and error handler.

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

```
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#endif
#include <syslog.h>
// 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 "tut_ex12.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex12.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(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++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
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_data(&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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        closelog();
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex12(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.14 tut_ex13_1.c

Defining a table with a data structure for the table.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex13_1.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex13_1.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex13_1(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.15 tut_ex13_2.c

Defining a table with a data structure for table and cells.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex13_2.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex13_2.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
//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;
//}
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="Workphone:",
                .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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex13_2(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

15.16 tut_ex14.c

Defining a table with widgets.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.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 "tut_ex14.pdf"
#else
#define OUTPUT_FILE "/tmp/tut_ex14.pdf"
#endif
#define TRUE 1
#define FALSE 0
// For simulated exception handling
jmp_buf env;
#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
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
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];
    time_t t = time(NULL);
    ctime_r(&t, buf);
    return buf;
}
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 = 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 = 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);
    // 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);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#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
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex14(pdf_doc, pdf_page);
    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

Index