

# Aculab Prosody™ API Guide

Fax Image Processing “ACTIFF” for Prosody DSP modules

## Proprietary Information

The information contained in this document is the property of Aculab plc., and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. It should not be used for commercial purposes without prior agreement in writing.

All trademarks recognised and acknowledged.

Aculab endeavours to ensure that the information in this document is correct, but does not accept liability for any error or omission.

The development of Aculab products and services is continuous and published information may not be the most recent version available. It is important to check the current position with Aculab.

© Aculab Plc. 2003: All rights reserved

## Revision Status

Revision		Comment
01i	24.06.99	Revised Issue to new format
02i	23.08.99	Extended to include Text and Add/remove
03i	08.09.99	Minor Typos
1.0r	09.01.00	Preparation for release 1.0.0
1.01r	18.02.00	Minor change to <code>actiff_save</code>
1.1r	20.03.01	Addition of new <code>actiff_bdf_font_init</code> api
1.2	28.03.01	Update to new format
2.0	19.09.03	Reviewed for Prosody 2 TiNG release
2.1	22.06.06	New mode ( <code>kACTIFFTextModeMerge</code> ) for <code>actiff_add_text_lines</code>
2.3	22.11.06	Routine maintenance of document content.
2.4	04.06.09	Updating document content

## Contents

1 Overview .....	4
2 ACTIFF file API .....	5
2.1 ACTIFF files.....	5
2.2 actiff_read_open.....	5
2.3 actiff_write_open .....	6
2.4 actiff_close.....	6
2.5 actiff_seek_page .....	7
2.6 actiff_page_properties .....	7
2.7 actiff_set_page_properties .....	10
2.8 actiff_save .....	10
3 TIFF page generation and combination.....	12
3.1 actiff_create .....	12
3.2 actiff_new_page .....	12
3.3 actiff_insert_actiff_pages.....	13
3.4 actiff_insert_tiff_pages.....	14
3.5 actiff_remove_pages .....	16
4 Headers, footers and text pages.....	17
4.1 actiff_add_text_lines.....	17
4.2 actiff_build_default_font.....	19
4.3 actiff_free_font.....	20
4.4 actiff_bdf_font_init .....	20

# 1 Overview

This document details the Aculab API used to manipulate images for use in fax transmission. It should be used in conjunction with the Fax Processing API guide.

Prosody's Fax transmission and reception algorithms require pre-formatted data which is encapsulated in an `ACTIFF_FILE` structure.

A group of `actiff` API calls allows the programmer to open TIFF files for reading or writing, and to manipulate the TIFF document while in memory. The properties of the TIFF images can be read and manipulated using this interface.

Aculab currently supports Group 3 TIFF files with 1-D coded (MH), 2-D coded (MR) or ITU-T T.6 (MMR) data.

The `actiff` API includes the following calls:

High level <code>actiff</code> Call	Description
<code>actiff_read_open</code>	Opens a TIFF file for reading (transmission).
<code>actiff_write_open</code>	Opens a TIFF file for writing (reception)
<code>actiff_close</code>	Closes a specified <code>ACTIFF_FILE</code> .
<code>actiff_page_properties</code>	Retrieves the properties of the current image.
<code>actiff_set_page_properties</code>	Sets the properties of the current page.
<code>actiff_seek_page</code>	Sets the current page to <i>page_number</i>
<code>actiff_create</code>	Creates an empty <code>ACTIFF_FILE</code> .
<code>actiff_new_page</code>	Creates a new empty page.
<code>actiff_save</code>	Save a structure to a new file.
<code>actiff_insert_actiff_pages</code>	Inserts a range of contiguous pages.
<code>actiff_insert_tiff_pages</code>	Inserts a range of contiguous pages.
<code>actiff_remove_pages</code>	Removes a range of contiguous pages.
<code>actiff_draw_text</code>	Not documented, please contact Aculab support
<code>actiff_build_default_font</code>	Returns the default font.
<code>actiff_free_font</code>	Frees the memory used by the font.
<code>actiff_bdf_font_init</code>	Opens a font file.
<code>actiff_add_text_lines</code>	Inserts up to two lines of text.
<code>actiff_draw_shapes</code>	Not documented, please contact Aculab support
<code>actiff_version</code>	Return the current version

## 2 ACTIFF file API

### 2.1 ACTIFF files

#### General

An `ACTIFF_FILE` is an internal representation of a fax document. When saved to disk it is a file in TIFF format and readable by ordinary TIFF imaging software. `ACTIFF_FILES` can be loaded from any Group 3 format TIFF file.

**Note** Group 3 TIFF is a small subset of the TIFF specification. `ACTIFF_FILES` can not be read from any other type of TIFF file.

#### Documents, Pages and Images

A general TIFF file comprises one or more images. Each image usually refers to the page of a document. Each image in the file can potentially be in a different format, such as Run-length coded, JPEG coded, or Group 3 fax coded. Also, each page of an image can have different properties, such as width, length, and resolution. `ACTIFF_FILES` will read in only Group 3 fax coded images.

#### Image File Directory – IFD

Each image in a TIFF file is described by an IFD. This encapsulates the page properties, i.e. image coding type, page length, resolution, software name, date-time, and the image data itself. The TIFF file contains a list of these IFDs, one for each page.

#### Image coding

`ACTIFF` can read and write Group 3 fax coded TIFF files in three formats:

1. 1-dimensional coded Modified Huffman (1D or MH)
2. 2-dimensional coded Modified Read (2-D or MR)
3. 2-dimensional Modified Modified Read (MMR), only used in Error Correction Mode.

**Note** TIFF files created by `ACTIFF` conform to ITU-T RFC2306, which restricts the definition of “Group 3 TIFF”

### 2.2 `actiff_read_open`

#### Prototype Definition

```
ACTIFF_FILE* actiff_read_open(const char* filename, int *perrno)
```

#### Parameters

*filename*

A pointer to a null-terminated string of ascii characters, containing the name of an existing TIFF file to be opened.

*perrno*

Upon failure this is a pointer to an integer. Will hold the system `errno` relating to opening the file.

#### Description

Opens an existing TIFF file for subsequent reading, for example by the Aculab

Prosody Fax API.

### Returns

The return value is a pointer to a valid `ACTIFF_FILE`, or `NULL` on error.

In the event of an error, the specific reason for error can be obtained from `*perrno`.

## 2.3 `actiff_write_open`

### Prototype Definition

```
ACTIFF_FILE* actiff_write_open(char* filename, ACTIFF_PAGE_PROPERTIES*
properties, int *perrno)
```

### Parameters

#### *filename*

A pointer to a null-terminated string of `ascii` characters, containing the name of the TIFF file to be created.

#### *properties*

Ignored. Included for backward compatibility only.

#### *perrno*

This is a pointer to an integer type that will hold the system `errno` relating to opening the file.

### Description

Opens a TIFF file for subsequent writing, for example, by the Aculab Prosody Fax API.

**CAUTION** If a file already exists with name *filename*, the function call will succeed but may lead to undefined behaviour.

### Returns

A pointer to a valid `ACTIFF_FILE`, or `NULL` on error is returned.

In the event of an error, the specific reason for the error can be obtained from `*perrno`.

## 2.4 `actiff_close`

### Prototype Definition

```
int actiff_close(ACTIFF_FILE* actiff, int *perrno)
```

### Parameters

#### *actiff*

An open `ACTIFF_FILE` pointer opened using `actiff_read_open` or `actiff_write_open`.

#### *perrno*

A pointer to an integer that will hold the system `errno` relating to closing the file.

### Description

Closes a specified `ACTIFF_FILE`, writes the TIFF header to the file, and frees up

memory used by the structure.

### Returns

Zero if successful.

`ERR_ACTIFF_INVALID` if the `actiff` pointer specified is null.

In the event of an error, the specific reason for error can be obtained from `*perrno`.

## 2.5 `actiff_seek_page`

### Prototype Definition

```
int actiff_seek_page(ACTIFF_FILE* actiff, int which)
```

### Parameters

#### *actiff*

An open `ACTIFF_FILE` pointer opened using `actiff_read_open` or `actiff_write_open`.

#### *page\_number*

The number of the page required, starting at zero.

### Description

Sets the current page to *which*, if it exists.

### Returns

Zero on success, Non-zero if the page number does not exist.

`ERR_SM_BAD_PARAMETER` if the page number is less than zero.

`ERR_ACTIFF_INVALID` if the `actiff` pointer specified is null.

## 2.6 `actiff_page_properties`

### Prototype Definition

```
int actiff_page_properties(ACTIFF_FILE *actiff,  
ACTIFF_PAGE_PROPERTIES* properties);
```

### Parameters

#### *actiff*

A pointer to an open `ACTIFF_FILE`.

#### *properties*

A pointer to an allocated structure of type `ACTIFF_PAGE_PROPERTIES`, see the description below.

### Description

This call retrieves the properties of the image within the current page of the `ACTIFF_FILE`. These properties would normally be used to modify the properties of a file being written, using a subsequent call to `actiff_set_page_properties`.

```
typedef struct _smtf_page_properties
```

```

{
    char    software_name[kACTIFFStringLength+1];
    char    document_name[kACTIFFStringLength+1];
    char    date_time[kACTIFFDateTimeLength+1];
    int     byte_aligned_eol;
    int     fill_order;
    int     image_coding;
    int     image_width;
    int     orientation;
    float   x_resolution;
    float   y_resolution;
    int     image_length;
    int     badfaxlines;
    int     consecutivebadfaxlines;
    int     cleanfaxdata;
} ACTIFF_PAGE_PROPERTIES;

```

***software\_name***

Name and version number of the software package(s) used to create the image.

***document\_name***

The name of the document from which this image was scanned.

***date\_time***

The format is: "YYYY:MM:DD HH:MM:SS", time is to be specified in 24 hour clock format. There must be one space character between the date and the time. The length of the string, including the terminating NULL, is (kACTIFFDateTimeLength) 20 bytes.

***byte\_aligned\_eol***

A flag indicating when T.4 data is padded to the byte boundary.

***fill\_order***

The logical order of bits within a byte, may be one of the following:

**FILL\_ORDER\_MSB\_FIRST** - Fill order 1 – codeword bits are arranged within a byte such that pixels with lower column values are stored in the higher-order bits of the byte. (Default)

**FILL\_ORDER\_LSB\_FIRST** - Fill order 2 - codeword bits are arranged within a byte such that pixels with lower column values are stored in the lower-order bits of the byte. Support for fill order 2 is not required in a Baseline TIFF compliant reader

**Note** Fill order 2 should be used only when `BitsPerSample` equals one and the data is either uncompressed or compressed using CCITT 1D or 2D compression, to avoid potentially ambiguous situations.

***image\_coding***

The image coding format may be one of the following values:

0	IMAGE_CODING_INVALID	invalid format
1	IMAGE_CODING_1D	1 dimensional coded modified Huffman

2	IMAGE_CODING_2D	2 dimensional coded modified read
4	IMAGE_CODING_MMR	2 dimensional modified modified read

***image\_width***

This field holds the number of pixels per scan line. Supported widths are 1728 pels and 2048 pels.

***orientation***

The orientation of the image with respect to the rows and columns and may be one of the following:

1. The 0th row represents the visual top of the image, and the 0th column represents the visual left-hand side.
2. The 0th row represents the visual top of the image, and the 0th column represents the visual right-hand side.
3. The 0th row represents the visual bottom of the image, and the 0th column represents the visual right-hand side.
4. The 0th row represents the visual bottom of the image, and the 0th column represents the visual left-hand side.
5. The 0th row represents the visual left-hand side of the image, and the 0th column represents the visual top.
6. The 0th row represents the visual right-hand side of the image, and the 0th column represents the visual top.
7. The 0th row represents the visual right-hand side of the image, and the 0th column represents the visual bottom.
8. The 0th row represents the visual left-hand side of the image, and the 0th column represents the visual bottom.

Default is 1 (top left).

**Note** Support for orientations other than 1 is not a Baseline TIFF requirement.

***x\_resolution***

The number of pixels per ResolutionUnit in the ImageWidth, (typically horizontal) direction.

***y\_resolution***

The number of pixels per ResolutionUnit in the ImageLength (typically, vertical) direction.

***resolution\_unit***

Applications often want to know the size of the picture represented by an image. This information can be calculated from the Image Width, Image Length, etc. and one of the following values:

1. No absolute unit of measurement. Used for images that may have a non-square aspect ratio but no meaningful absolute dimensions.
2. Inch, (Default).
3. Centimeter.

***image\_length***

The number of rows (sometimes described as *scanlines*) in the image.

***badfaxlines***

This fields keeps a record of the total number of scan lines received in a corrupted state.

***consecutivebadfaxlines***

In an image containing bad lines, there may be a number of consecutive bad lines. This member keeps a record of the largest number of consecutive lines.

**Note** As the fax processing algorithms may be able to fix some bad lines, the values of `badfaxlines` and `consecutivebadfaxlines` may not reflect the actual number of bad lines and consecutive bad lines in the end image.

***cleanfaxdata***

May be one of the following:

<code>CLEANFAXDATA_CLEAN</code>	There are no errors in the fax data.
<code>CLEANFAXDATA_REGENERATED</code>	Errors have been corrected by regenerating lines.
<code>CLEANFAXDATA_UNCLEAN</code>	There are still errors in the fax data

**Returns**

Zero on success

`ERR_ACTIFF_INVALID` the specified `actiff` pointer is null.

## 2.7 `actiff_set_page_properties`

**Prototype Definition**

```
int actiff_set_page_properties(ACTIFF_FILE *actiff,
ACTIFF_PAGE_PROPERTIES* properties);
```

**Parameters*****actiff***

A pointer to an open `ACTIFF_FILE`.

***properties***

A pointer to an allocated structure of type `ACTIFF_PAGE_PROPERTIES`, containing the new properties to be written to the current page.

**Description**

Sets the properties of the current page of the `ACTIFF_FILE`. These properties would be modified from properties read using a prior call to `actiff_page_properties`.

**Returns**

Zero on success

Non-zero if the specified properties are inappropriate for Group 3 Fax as understood by `actiff`.

## 2.8 `actiff_save`

**Prototype Definition**

```
int actiff_save(ACTIFF_FILE *actiff, char *filename, int *perrno);
```

**Parameters*****actiff***

A pointer to an open `ACTIFF_FILE`.

***filename***

A pointer to a null-terminated string of `ascii` characters, containing the name of the file to be saved.

***perrno***

This is a pointer to an integer that will hold the system `errno` relating to closing the file.

**Description**

It is possible to modify an `ACTIFF_FILE` structure while in memory. `actiff_save` allows an application to save the modified structure to a new file on disk with the name `filename`. This is useful if images have been modified using `actiff_add_text_lines`, or if the page ordering has changed.

**Note** This function is not used during fax reception. `ACTIFF` files created using `actiff_write_open`, and submitted to the Aculab Prosody Fax library for fax reception will be written to disk during the fax reception process. They should be closed with `actiff_close`. This function can be used for `ACTIFF` files that were created using `actiff_read_open` (possibly modified subsequently) and `ACTIFF` files that were created using `actiff_create`. In the former case, the filename must be different from that submitted to `actiff_read_open`, otherwise `ERR_SM_FILE_ACCESS` will result.

**Returns**

Zero on success.

<code>ERR_SM_FILE_ACCESS</code>	if the file cannot be opened for writing.
<code>ERR_ACTIFF_INVALID</code>	if <code>actiff</code> is invalid,
<code>ERR_ACTIFF_PAGE_RANGE</code>	if <code>actiff</code> has no pages

On return from `actiff_save`, the current page will be zero (the first page). Use `actiff_seek_page` to move between pages.

## 3 TIFF page generation and combination

### 3.1 actiff\_create

#### Prototype Definition

```
ACTIFF_FILE* actiff_create( void )
```

#### Description

Creates an `ACTIFF_FILE`, which is not associated with a TIFF file. The `ACTIFF_FILE` holds no pages, and no images. The `ACTIFF_FILE` cannot be used at this stage for reading or writing data. Pages can be added using `actiff_new_page`, `actiff_add_actiff_pages` or `actiff_add_tiff_pages`.

#### Returns

The function returns a pointer to the newly created `ACTIFF_FILE` if the call was successful.

NULL if the `ACTIFF_FILE` could not be created.

### 3.2 actiff\_new\_page

#### Prototype Definition

```
int actiff_new_page( struct actiff_new_page_parms* create_parms)
```

#### Parameters

##### ***create\_parms***

A pointer to a structure of the following type:

```
typedef struct actiff_new_page_parms
{
    ACTIFF_FILE*          actiff;
    int                   before_target_page;
    ACTIFF_PAGE_PROPERTIES properties;
} ACTIFF_NEW_PAGE_PARMS;
```

#### Description

Creates a new empty page within the `ACTIFF` document.

##### ***actiff***

Should point to a valid `ACTIFF_FILE` into which the new page should be inserted. Typically, this pointer would have been returned by `actiff_create` or `actiff_write_open`.

##### ***before\_target\_page***

Denotes the page number before which the new page will be inserted. This will be the page number of the new page. This should be less than or equal to the number of pages in `actiff`. For the purposes of `ACTIFF_FILES`, pages are numbered from zero, therefore if `page_number` is zero, a new first page will be inserted into `ACTIFF_FILE` before the existing first page.

If `before_target_page` is assigned `kACTIFFPageAppend`, or if it is higher than the

number of existing pages, the new page will be appended to the end of `ACTIFF_FILE`.

#### ***properties***

If `properties.image_length` is zero, the new page contains no images.

If `properties.image_length` is non-zero, the new page will contain an image, which is entirely white, and has this number of lines. Blank lines can also be added using `actiff_add_text_lines`.

**Note** If the `ACTIFF_PAGE_PROPERTIES` structure contains properties that are not compliant with Group 3 Fax (e.g. `image_coding=255`) this function will fail with `ERR_ACTIFF_INVALID`.

#### **Returns**

Zero on success

`ERR_ACTIFF_INVALID`      `actiff` was invalid or not open.

`ERR_ACTIFF_PAGE_RANGE`      The page number was outside the page range.

If the function is successful, the current `actiff` page will be the new page.

### 3.3 `actiff_insert_actiff_pages`

#### **Prototype Definition**

```
int actiff_insert_actiff_pages( struct actiff_insert_pages_parms*
insert_parms )
```

#### **Parameters**

`insert_parms` is a pointer to a structure of the following type:

```
typedef struct actiff_insert_pages_parms
{
    ACTIFF_FILE      *actiff;
    ACTIFF_FILE      *source_actiff;
    char              *filename;
    int               source_first_page;
    int               source_num_pages;
    int               before_target_page;
    int               err;
} ACTIFF_INSERT_PAGES_PARMS;
```

#### **Description**

Takes `source_num_pages` contiguous pages, starting at page index `source_first_page` from an open source (`source_actiff`) and inserts them into an existing destination (`actiff`) at a position before `before_target_page`.

#### **Parameters**

***actiff***

Must hold a pointer to an already open `ACTIFF_FILE`, preferably one that was returned by `actiff_create` or `actiff_write_open`. This is the destination structure.

***source\_actiff***

Must hold a pointer to an already open `ACTIFF_FILE`. This would be a pointer returned by `actiff_read_open`.

*filename*

Ignored.

*source\_first\_page*

This page index identifies where to begin the copying process.

**Note** Page indexing begins at 0.

*source\_num\_pages*

Identifies the number of pages to insert.

Set this parameter to `kACTIFFPagesAll` if all pages following `source_first_page` from `source_actiff` are to be inserted.

*before\_target\_page*

Denotes the page number before which the new pages will be inserted. This will become the page number of the first new page. This should be less than or equal to the number of pages in `actiff`. For the purposes of the `ACTIFF_FILE` pages are numbered from zero, therefore if `page_number` is zero a new first page will be inserted into `actiff` before the existing first page.

If `before_target_page` is assigned `kACTIFFPageAppend`, or if it is higher than the number of existing pages, the new page will be appended to the end of `actiff`.

*err*

Ignored. Deprecated and may be removed in future releases.

Returns

Zero on success

`ERR_ACTIFF_INVALID` `actiff` or `source_actiff` is invalid or not open

`ERR_ACTIFF_PAGE_RANGE` `source_actiff` does not contain the page range requested.

On success, the current page is the last of the inserted pages.

### 3.4 `actiff_insert_tiff_pages`

Prototype Definition

```
int actiff_insert_tiff_pages ( struct actiff_insert_pages_parms*
insert_parms )
```

Parameters

`insert_parms` is a pointer to a structure of the following type:

```
typedef struct actiff_insert_pages_parms
{
    ACTIFF_FILE    *actiff;
    ACTIFF_FILE    *source_actiff;
    char           *filename;
    int            source_first_page;
    int            source_num_pages;
}
```

```

    int          before_target_page;
    int          err;
} ACTIFF_INSERT_PAGES_PARMS;

```

### Description

Opens a Group 3 TIFF file described by `filename` and inserts pages from this TIFF file into an open `ACTIFF_FILE`.

### Parameters

#### *actiff*

Must hold a pointer to an already open `ACTIFF_FILE`. This is the target.

#### *source\_actiff*

This is a pointer to an already open `ACTIFF_FILE` which contains images. Or can be NULL if `filename` is specified.

#### *filename*

The name of the TIFF file that is to be inserted into the destination, `actiff`. Ignored if `source_actiff` points a valid `ACTIFF_FILE`, opened for reading.

#### *source\_first\_page*

Identifies the first page to be inserted.

**Note** page indexing starts at 0

#### *source\_num\_pages*

Identifies the number of pages to insert.

Set this parameter to `kACTIFFPagesAll` if the entire TIFF image is to be inserted.

#### *before\_target\_page*

Denotes the page number before which the new page will be inserted. This will become the page number of the first new page. This should be less than or equal to the number of pages in `actiff`. For the purposes of the `ACTIFF_FILE` pages are numbered from zero, therefore if `page_number` is zero a new first page will be inserted into `actiff` before the existing first page.

If `before_target_page` is assigned `kACTIFFPageAppend`, or if it is higher than the number of existing pages, the new page will be appended to the end of `actiff`.

#### *err*

Ignored. Deprecated and may be removed in future releases.

### Returns

The function returns 0 if the call was successful, or one of the following error codes

`ERR_ACTIFF_FILE_ERROR` `filename` does not describe a valid Group 3 TIFF file.

`ERR_ACTIFF_INVALID` `actiff` or `source_actiff` is invalid or not open

`ERR_ACTIFF_PAGE_RANGE` `source_actiff` does not contain the page range requested.

`ERR_SM_BAD_PARAMETER` if `source_actiff` and `filename` are both specified. Also returned if `source_actiff` and `filename` are both not specified.

On return, the current page of `actiff` is the last of the inserted pages.

### 3.5 actiff\_remove\_pages

#### Prototype Definition

```
int actiff_remove_pages ( struct actiff_remove_pages_parms*
remove_parms )
```

#### Parameters

`remove_parms` is a pointer to a structure of the following type:

```
typedef struct actiff_remove_pages_parms
{
    ACTIFF_FILE*    actiff;
    int             first_page;
    int             num_pages;
} ACTIFF_REMOVE_PAGES_PARMS;
```

#### Description

Removes a range of contiguous pages from an open `ACTIFF_FILE`.

#### Parameters

***actiff***

Must hold a pointer to an already open `ACTIFF_FILE`.

***first\_page***

Identifies the first page to be removed from `actiff`.

**Note** The first page is indexed from page 0

***num\_pages***

Identifies the number of consecutive pages, beginning at `first_page`, to be removed from `actiff`. If this is set to `kACTIFFPagesAll` then all pages, starting with the specified first page, will be removed.

#### Returns

The function returns 0 if the call was successful, or one of the following error codes

<code>ERR_ACTIFF_INVALID</code>	<code>actiff</code> is invalid or not open
<code>ERR_ACTIFF_PAGE_RANGE</code>	<code>actiff</code> does not contain the page range requested.

## 4 Headers, footers and text pages

### Text Insertion

A limited facility for insertion of text into images is provided by `ACTIFF`. Font support is very limited and is intended for addition of headers and footers, which generally give information about time, date, station ID and such like. It is possible to generate text-only pages by creating a blank page, and adding text lines.

**Note** It is conventional for fax-sending system to provide a header, and for the receiving machine to provide a footer.

### 4.1 `actiff_add_text_lines`

#### Prototype Definition

```
int actiff_add_text_lines ( struct actiff_add_text_lines_parms*
text_parms )
```

#### Parameters

`text_parms` is a pointer to a structure of the following type:

```
typedef struct actiff_add_text_lines_parms
{
    ACTIFF_FILE          *actiff;
    int                  text_mode;
    ACTIFF_TEXT_LINE     line1;
    ACTIFF_TEXT_LINE     line2;
    ACTIFF_FONT          *font;
    float                 page_length;
    int                   page_length_type;
} ACTIFF_ADD_TEXT_LINES_PARMS;

typedef struct actiff_text_line
{
    char                  *left_text;
    char                  *centre_text;
    char                  *right_text;
    float                 position;
    float                 margin;
    int                   position_type;
} ACTIFF_TEXT_LINE;
```

#### Description

Inserts up to two lines of text into the current page of an open `ACTIFF_FILE`.

It is required that `actiff_seek_page` be used first to navigate to the page that is to hold the text specified in the `ACTIFF_TEXT_LINE`.

#### Parameters

***actiff***

This is a pointer to an `ACTIFF_FILE` that holds the page to which text is to be added.

***text\_mode***

Has three possible values:

`kACTIFFTextModeInsert`, which inserts text lines into the existing image. Existing image lines remain intact and consequently the page length increases by the combined height of the lines of text. The height of each line of text is defined by the `num_lines` element of the `ACTIFF_FONT` structure.

`kACTIFFTextModeReplace` maintains the page length by over-writing image lines with text.

`kACTIFFTextModeMerge` maintains the page length by merging text lines with the existing image. The resulting image is similar to that produced by using `kACTIFFTextModeReplace`, however, in this mode existing content on the affected lines is preserved.

***font***

A pointer to a structure of type `ACTIFF_FONT`, which is returned by the function `actiff_build_default_font` or `actiff_bdf_font_init`.

**Note** Currently this font only supports 8-bit, printable ASCII characters. Character values outside of this range will be replaced by a white-space character.

***page\_length***

Specifies the required (vertical) length of the page, after the text has been added. This can be used to extend the length in parallel with adding text. It may also be necessary in order to give meaning to dimensions, which are given relative to the bottom of the page, e.g. `kACTIFFInchesFromBottom`. If `page_length` is set to zero, the parameter is ignored.

***page\_length\_type***

Specifies the dimensions used in the `page_length` parameter. There are 3 possible values.

`kACTIFFInchesFromTop`, page length in terms of inches.

`kACTIFFCMFromTop`, page length in terms of centimetres.

`kACTIFFLinesFromTop` page length in terms of image (pixel) lines.

***line1 and line2***

Describe the text to be added, in the form of an `ACTIFF_TEXT_LINE` structure, the structure is described below:

***left\_text, centre\_text and right\_text***

Three optional text strings. A single line of text will be formatted using these three strings as follows:

`left_text` is left-justified.

`right_text` is right-justified.

`centre_text` is centre-justified.

In the event that the total length is wider than the page, the text line will be truncated. The individual strings have the following priority:

1 `centre_text` is moved to the left or right in order to avoid `left_text` and

right\_text.

2 centre\_text is truncated to fit between left\_text and right\_text

3 right\_text is truncated to fit to the right of left\_text

4 left\_text is truncated to fit on the page.

If all of the text fields are NULL, no line will be added, even in "insert" mode.

**position**

Specifies the vertical position of the line of text, relative to the top or bottom of the page, as specified by position\_type.

**position\_type**

Can have one of the following values, which are:

kACTIFFLinesFromTop	pixel lines from top of page
kACTIFFLinesFromBottom	pixel lines from foot of page
kACTIFFInchesFromTop	inches from top of page
kACTIFFInchesFromBottom	inches from bottom of page
kACTIFFCMFromTop	centimetres from top of page
kACTIFFCMFromBottom	centimetres from bottom of page

Positions relative to the bottom of the page specify the bottom of the text line. Positions relative to the top of the page specify the top of the text line. Bear in mind that in "insert" mode, the page length will alter when the lines are inserted.

In the event that the position of line2 is higher up the page than that of line1, the position of the second line will be adjusted to just below that of the first. That is, it is a constraint of this function that line2 is below line1.

**margin**

specifies a margin to the left and right of the text line. The units of margin are interpreted as consistent with the position\_type.

**Returns**

Zero if successful.

The function returns 0 if the call was successful, or one of the following error codes

ERR_ACTIFF_INVALID	actiff is invalid or not open.
ERR_SM_OS_RESOURCE_PROBLEM	in the event that internal memory allocations were not possible.

## 4.2 actiff\_build\_default\_font

**Prototype Definition**

ACTIFF\_FONT \*actiff\_build\_default\_font(void)

**Description**

Returns the default ACTIFF\_FONT for use in text-insertion routines.

**Returns**

The function returns a pointer to the ACTIFF\_FONT object if the call was successful, or NULL if the font could not be located.

### 4.3 actiff\_free\_font

#### Prototype Definition

```
void actiff_free_font(ACTIFF_FONT *font)
```

#### Description

Frees the memory used by `font`. The parameter `font` would have typically been returned by `actiff_build_default_font` or `actiff_bdf_font_init`.

### 4.4 actiff\_bdf\_font\_init

#### Prototype Definition

```
ACTIFF_FONT *actiff_bdf_font_init (char *bdfFileName, int *ErrorCode)
```

#### Description

This function opens the font file specified by `"bdfFileName"` for reading in text mode. It is assumed that the BDF font format is version 2.1 or 2.2. The bitmap data is extracted from the font file and converted into an `ACTIFF_FONT` structure.

**Note** This API only handles text versions of BDF font files. Binary format BDF font files are not supported.

**Note** Unicode variants of BDF font files are not supported.

It is acknowledged that BDF font files can contain more than 256 characters (glyphs). In such cases only the first 256 glyphs are encoded. The remaining glyphs are ignored. Therefore, only the first 256 characters will be available for use.

#### Returns

Upon success, `actiff_bdf_font_init` returns a pointer to a valid `ACTIFF_FONT` structure. This pointer can be used as required. `NULL` is returned otherwise, and `ErrorCode` is set to a non-zero value and may be one of the following:

<code>ERR_TAG_UNPAIRED</code>	Encountered an odd number of tags in specified font file.
<code>ERR_TAG_GLYPH_L</code>	Font file contains fewer Glyphs than expected.
<code>ERR_TAG_GLYPH_G</code>	Font file contains more Glyphs than expected.
<code>ERR_BADFILE</code>	The specified file maybe corrupt or may not exist.
<code>ERR_MEMALLOC</code>	Necessary resources could not be allocated.
<code>ERR_TOO_MANY_CHARS</code>	Font file contains more than 256 glyphs.

**NB:** All the above error codes cause the function to return a `NULL` pointer. Except `ERR_TOO_MANY_CHARS`, this error code is not deemed as critical, only the first 256 glyphs are encoded and a valid pointer is returned in such cases.