ARCLIB

Archos Media Library Specification

AGREEMENT OF USAGE

This document is for information purposes only and does not constitute any agreement between Archos SA or its subsidiaries and you other than descibed here below. This document may only be reproduced in its entirety and must include this AGREEMENT OF USAGE message.

Archos does not guarantee the accuracy of this information and usage of this information and any results of it are at the complete responsibility of the end user. Archos does not provide any user support for this information. The results of any work derived from usage of this information which render your Archos device unusable or unsatisfactory in performance are assumed by the creator of such work.

ALL RISKS AND LIABILITIES INVOLVED IN USING THIS INFORMATION FOR USAGE WITH ARCHOS PRODUCTS IS ASSUMED BY THE END USER.

Revision history:

v1.0	**10.04.2003**	- **initial version**
v1.1	**29.04.2003**	- **added private data section in header** - **added requirement for a "search list" in the library structure** - **added year and genre information for media files**
V1.2	**02.05.2003**	- **added chunk header for private data** - **added flags to file_str** - **added WMA type**
V1.3	**05.05.2003**	- **added UTF8 encoding for strings** - **added byte order definition** - **changed alignment for all sections to hard drive sector size** - **added genre list** - **removed sample library file**
V1.4	**02.07.2003**	- **append list_entries definition** - **append data types definition**
V1.5	**18.05.2004**	- **add max library size** - **add filename and position** - **add libdump.c source code**

Archos Media Library Specification

Introduction:

Archos MP3 and Multimedia products use no proprietary file system on their hard drives. Files can be up and downloaded from and to the units with standard OS tools (e.g Windows Explorer) using a generic USB mass storage driver. The firmware on the Archos players can "browse" the hard drive and play the music or multimedia data. "Organizing" the media content on the players is left to the user (e.g. put music files in a /artist/album/title folder hierarchy), Archos players do not impose a certain format or folder layout on the hard drive.

Besides the file name (and the location on the hard drive) media files can have a variety of additional "meta" information associated with them (e.g. information about artist/album/composer/year etc. found in ID3 tags). Users want to "browse" or search their media files on the player according to this meta information.

To allow for meta information based browsing Archos is implementing a "media library" feature for its players. The media library is a special file on the hard drive which holds structured information about the media files on the hard drive. If present, the media library is read by the players and used to offer meta information based browsing in addition to browsing the hard drive contents directly.

The format of the media library file is open to interested parties so that they can compile a media library for Archos players from their own databases.

Scope:

This document describes only the “ARCLIB” file format as it is used on Archos GMINI120 and GMINI220 products, the information may not be valid for future Archos products.

Media Library structure:

The media library has two basic types of data, "files" and "lists":

- "Files" are music content, like MP3/WMA/WAV songs.
- "Lists" are used to organize and structure the "files" for easy access by the user. A "list" is just a collection of further "lists" or "files", comparable to a folder on a hard drive.

The structure defined by "lists" and "files" can by arbitrary (within certain limits), there is no imperative on how the files are to be organized into folders.

Lists can contain only further lists or only files, mixed lists that have both, lists and files are not allowed.

List also need to be unique, it is not allowed to "link" an existing list in a later defined one. This ensures that the tree structure is indeed a tree and for each list a unique "parent" list can be found.

There must exist at least one list which holds all media content and therefore can be used for searching. This list can be included in the list structure (e.g. "All Songs") but it can also be a "hidden" list which will not be reachable from the "root" list. The number of this list must be given in the library header.

Media content inside a list should be sorted in a reasonable way (e.g. alphabetically for artists/allbums etc. or according to track number for songs from one album)

Media content like songs, video etc can be referred to in multiple lists, this allows to structure the same content by different criteria.

Playlists (which on the PC would be a M3U file or similar) are also created using lists in the library, the actual .m3u files which may be on the hard drive are not referenced in the library.

Example structure of a library file:

List 1: Root
#1: Artists
#2: Albums
#3: Songs
#4 Playlists

List 2: Artists
#1: Beatles
#2: Stones

List 3: Beatles
#1: White CD
#2: Black CD

List 4: White CD
#1: Help.mp3
#...

List 5: Black CD
#1: Black.mp3
#...

List 6: Stones
#1: Best of

List 7: Best of
#1: we_rock.mp3
.....

List 8: Albums
#1: Best of
#2: Black CD
#2: White CD

List 9: Best of
#1: we_rock.mp3
.....

List 10: Black CD
#1: Black.mp3
#...

List 11: White CD
#1: Help.mp3
#...

List 12: Songs
#1: Black.mp3
#....
#...: Help.mp3
#.....
#..: we_rock.mp3

List 13: Playlists
#1: Favorites

List: 14: Favorites
#1: Help.mp3
#2: Black.mp3
#3: we_rock,mp3

As seen there is a total of 14 lists which contain either further lists or MP3 songs.

The sample library is very simple and offers four diffferent “approaches” to the library content:

- **Artists** / All Artists / All Albums of one artist / All songs of one album of one artist
- **Albums** / All Albums of all artists / All Songs of one album
- **Songs** / All Songs off all artists
- **Playlists** / All Playlists / All Songs of one playlist

The "search list" is list #12, this list holds all songs

Additionally one could set up the library to include:

- **Genre** / All Genres / All Artist of one Genre / Albums … / Songs …..
- **Year** / All Year or periods (80s, 90s etc.) / All Artist from one year/period / Albums… / Songs…
- **Mood** / All Moods / All Songs from one mood …..

Media Library layout:

The library format is intended to have a very small memory footprint. It is not intended to be used as a data base, it is read only. All structures defined here assume packed little-endian data.

The library consists of these sections:

1. Header	struct head_str head
2. Files with *num_files* items	struct files_str files[num_files]
3. Lists with *num_list* items	struct list_str lists[num_lists]
4. List entries for all lists	uint16_t list_entries[...]
5. File path information for all files	uint8_t paths[*total paths size*]
6. String data for files, lists, paths	uint8_t strings[t*otal string size*]
7. Private data – if present	arbitrary structure and size

In order to keep the size of the library file small so that it can be held completely in RAM of the players, the number of files and lists (not list entries) together is limited to 65536 (16 bit).

The maximum size of the library file is 1MB for Gmini220 and 2MB for Gmini120 products.

"Files" and "Lists" share the same numbering, "lists" are numbered from 0 to (*num_list* – 1) and "lists" are numbered from *num_list* to (*num_list* + *num_file* – 1)

For the following structures these data types are used:

string_t:
    uint32_t, byte offset into strings[]
path_t:
    uint32_t, byte offset into paths[]
file_list_t:
    uint16_t, number of file or list, used as index into *lists* or *files* array.

All sections must be aligned to the next 512 byte (Hard drive sector size) boundary.

All structures must be “packed”, 8bit and 16bit size values must be packed into 32bit words.

All data is stored in “little endian” mode

1. Header

```
struct hdr_str {
        uint8_t      magic[4];              = "JBML"
        uint32_t     version;               = 0x00000101
        uint32_t     num_files;
        uint32_t     num_lists;
        uint32_t     offset_files;
        uint32_t     offset_lists;
        uint32_t     offset_list_entries;
        uint32_t     offset_paths;
        uint32_t     offset_strings;
        uint32_t     offset_private_data;
        uint32_t     search_list;
        uint8_t      res[468];
}
```

num_files, *num_lists* is the number of files and lists respectively. The sum of these two must be less than 65536.

offset_files, and *offset_lists* is the offset in bytes from the start of the file to the respective structure holding information about files and lists.

offset_list_entries is the offset in bytes from the start of the file to the variable length structure holding the individual list entries.

offset_paths is the offset in bytes from the start of the file to the variable length structures which hold the paths to the files.

offset_strings is the offset in bytes from the start of the file to the string pool, where all text strings used in files, lists and paths stored.

offset_private_data is the offset in bytes from the start of the file to additional private data. The contents of the data section are not specified and will not be used by Archos multimedia products. Private data must be located at the end of the library file. If there is no private data present, the offset should equal the file size and therefore point to the end of the file.

search_list gives the number of the list which will be used for searching through all content.

res fills the header structure to 512 bytes total length (for future expansion).

2. Files with *num_file* items

The file structure holds all information of one file in the media library and points to an actual file on the hard drive:

```
struct file_str {
        path_t      path;       pointer to the file path
        string_t    name;       file name on hard drive
        string_t    artist;     artist name (for music)
        string_t    album;      album name (for music)
        string_t    title;      song title (for music)
        uint8_t     flags;      flags to be used by Device, must be set to 0!
        uint8_t     track;      track # (on CD, record, for music)
        uint8_t     type;       file type
        uint8_t     genre;      file genre
        uint16_t    year;       file year
        uint16_t    reserved;   must be set to 0!
}
```

If a file has no *path* (i.e. its in the root folder of the hard drive) the path value is "–1"
If a text string for *artist*, *album* or *title* is not set, the value of the pointer is "-1"

Flags are used by the Multimedia Device during operation and must be set to 0 when the library is created.

The *track* # can be used to sort the songs in a list not only alphabetically but also according to the actual track sequence on the CD. The track # is limited to a maximum 255!

Currently these values for *type* are defined:

0	MP3 file	".mp3"
1	MP2 file	".mp2"
2	WAV-PCM file	".wav"
3	WMA file	“.wma”

As the actual filename extensions are not stored in the library it is imperative that each media file has the correct type set in order to be able to find it on the hard drive.

genre specifies the music genre according to the *extended Winamp genre list* as specified in appendix A.

year gives the recording/creation year.

Reserved is reserved for future use and should be set to 0!

3. Lists with *num_list* items

```
struct list_str {
        uint8_t         type;           list type
        uint24_t        offset;         pointer to list entries
        uint16_t        length;         number of list entries
        file_list_t     parent:         pointer to parent list
        string_t        name;           name of list
}
```

Currently these values for *type* are defined:

0	LIST_ROOT	(root list is always first list in library)
1	LIST_ARTIST	
2	LIST_ALBUM	
3	LIST_SONG	
4	LIST_M3U	
5	LIST_GENRE	
6	LIST_YEAR	

The list type is used by the player SW GUI to show different "icons" for different types of lists

If the list is the "search list" as indicated in the file header and the list is "hidden" (not accessible from list structure) then the root list should be given as *parent*.

4. List entries for all lists

As the number of files and lists is limited to 16bit, the actual list entries are just an array of unit16_t. The *offset* pointer of *struct list_str* gives the position of the first list entry in this array.

List entries are either references to further lists or media files. In case of lists, the list entries give the number of the list (*num_list* to (*num_list* + *num_file* – 1)), in case of files, the list entries give the index number (*0* to (*num_list* – 1)) of the files defined in *struct files_str files[].*

5. File path information for all files

In order to minimize the library size, all paths that give the actual location of files on the hard drive are stored in a special way. The path names are split into the individual folder names and these are stored in the string pool together with all other text strings. This way a lot of text strings can be "re-used" for multiple paths.

```
struct path_str {
        uint_32t        length;         length of the path (number of subfolders)
        string_t        folder[];       pointers to names of subfolders
}
```

Example:

/Music/Beatles/BestOf/Help.mp3

length = 3
folder[0] = pointer to string "Music"
folder[1] = pointer to string "Beatles"
folder[2] = pointer to string "BestOf"

6. String data for files, lists, paths

All text strings used in the library are stored as zero-terminated UTF8 strings. Strings are stored consecutively and are referenced by an offset from the start of the string data. All strings should be unique in order to minimize the library file size.

7. Private data

Private data if present consists of one or more data “chunks”. Each chunk is identified by a chunk header which holds a “*magic*” identifier and the *offset_next* pointer which holds the offset of the next chunk from the start of the library file. If there is no next chunk *offset_next* should equal the file size and therefore point to the end of the file.

```
struct chunk_hdr_str {
        uint8_t         magic[4];                = "CHNK"
        uint32_t        offset_next;
        uint8_t         data[ length_of_chunk ];
}
```

8. Filename

The library file must have the file name “lib.jbm” and it must be placed in the root directory of the player in order to be accessed.

Appendix A: Extended Winamp Genre List

0: Blues
1: Classic Rock
2: Country
3: Dance
4: Disco
5: Funk
6: Grunge
7: Hip-Hop
8: Jazz
9: Metal
10: New Age
11: Oldies
12: Other
13: Pop
14: R&B
15: Rap
16: Reggae
17: Rock
18: Techno
19: Industrial
20: Alternative
21: Ska
22: Death Metal
23: Pranks
24: Soundtrack
25: Euro-Techno
26: Ambient
27: Trip-Hop
28: Vocal
29: Jazz+Funk
30: Fusion
31: Trance
32: Classical
33: Instrumental
34: Acid
35: House
36: Game
37: Sound Clip
38: Gospel
39: Noise
40: Alternative Rock
41: Bass
42: Soul
43: Punk
44: Space
45: Meditative
46: Instrumental Pop
47: Instrumental Rock
48: Ethnic
49: Gothic
50: Darkwave
51: Techno-Industrial
52: Electronic
53: Pop-Folk
54: Eurodance
55: Dream
56: Southern Rock
57: Comedy
58: Cult
59: Gangsta
60: Top 40
61: Christian Rap
62: Pop/Funk
63: Jungle
64: Native American
65: Cabaret
66: New Wave
67: Psychadelic
68: Rave
69: Showtunes
70: Trailer
71: Lo-Fi
72: Tribal
73: Acid Punk
74: Acid Jazz
75: Polka
76: Retro
77: Musical
78: Rock & Roll
79: Hard Rock
80: Folk
81: Folk-Rock
82: National Folk
83: Swing
84: Fast Fusion
85: Bebob
86: Latin
87: Revival
88: Celtic
89: Bluegrass
90: Avantgarde
91: Gothic Rock
92: Progressive Rock
93: Psychedelic Rock
94: Symphonic Rock
95: Slow Rock
96: Big Band
97: Chorus
98: Easy Listening
99: Acoustic
100: Humour
101: Speech
102: Chanson
103: Opera
104: Chamber Music
105: Sonata
106: Symphony
107: Booty Bass
108: Primus
109: Porn Groove
110: Satire
111: Slow Jam
112: Club
113: Tango
114: Samba
115: Folklore
116: Ballad
117: Power Ballad
118: Rhythmic Soul
119: Freestyle
120: Duet
121: Punk Rock
122: Drum Solo
123: Acapella
124: Euro-House
125: Dance Hall
126: Goa
127: Drum & Bass
128: Club-House
129: Hardcore
130: Terror
131: Indie
132: BritPop
133: Negerpunk
134: Polsk Punk
135: Beat
136: Christian Gangsta Rap
137: Heavy Metal
138: Black Metal
139: Crossover
140: Contemporary Christian
141: Christian Rock
142: Merengue
143: Salsa
144: Trash Metal
145: Anime
146: Jpop
147: Synthpop

Appendix B: libdump.c source code

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

struct hdr_str {
	char		magic[4];
	unsigned int	version;
	unsigned int	num_files;
	unsigned int	num_lists;
	unsigned int	offset_files;
	unsigned int	offset_list;
	unsigned int	offset_list_entries;
	unsigned int	offset_paths;
	unsigned int	offset_strings;
	unsigned int	offset_private_data;
	unsigned int	search_list;
	char		reserved[468];
};

struct file_str {
	int		path;
	int		name;
	int		artist;
	int		album;
	int		title;
	unsigned int	flags		:8;
	unsigned int	track		:8;
	unsigned int	type		:8;
	unsigned int	genre		:8;
	unsigned int	year		:16;
	unsigned int	reserved :16;
};

struct list_str {
	unsigned int	type:	8;
	unsigned int	offset: 24;
	unsigned int	length: 16;
	unsigned int	parent: 16;
	unsigned int	name;
};

#define MAX_DEPTH 10
struct lib_path_str {
	int		length;
	int		path[MAX_DEPTH];
};

// buffer to store the list data
#define LIB_BUF_MAX	1024*1024
unsigned char lib_buf[LIB_BUF_MAX];

struct file_str		*files;
int			file_num = 0;

struct list_str		*lists;
int			list_num = 0;

unsigned short int	*list_buffer;
char			*strings;
```

```
char                    *paths;

int                     search_list;

static void utf8_to_latin1( unsigned char* src, char *dst )
{
        int utf8;

        while ( *src != '\0' ) {
                if ( *src < 0x80 ) {
                        // normal ASCII, just copy char
                        *dst++ = *src++;
                } else {
                        // utf-8 char
                        if ( ( *src >= 0xC0 ) && ( *(src+1) >= 0x80 ) ) {
                                // we have a 2-byte UTF-8 char
                                utf8 = ((*src & 0x1F) << 6) + (*(src+1) & 0x3F);
                                if ( utf8 < 0x100 ) {
                                        // latin1
                                        *dst++ = utf8;
                                }
                                // advance
                                src += 2;
                        } else {
                                // ignore all other
                                src++;
                        }
                }
        }
        *dst = '\0';
}

static char *get_string( int p )
{
        static char lib_string[256];
        utf8_to_latin1( (unsigned char*) (strings + p), lib_string );
        return lib_string;
}

static struct lib_path_str *get_path( int p )
{
        return (struct lib_path_str *) (paths + p);
}

static char *path_to_string( char *path, struct file_str *f )
{
        int i;
        char *p = path;

        strcpy( p, "");
        if ( f->path == -1 ) {
        } else {
                for ( i = 0; i < get_path(f->path)->length; i++ ) {
                        strcat( p, "/");
                        strcat( p, get_string(get_path(f->path)->path[i]) );
                }
        }
        strcat( p, "/");
        strcat( p, get_string( f->name ) );

        return p;
}
```

```
static struct list_str *LIB_GetList( int list )
{
	return lists + (list - file_num);
}

static struct file_str *LIB_GetFile( int file )
{
	return files + file;
}

static int LIB_GetListItem( struct list_str *list, int pos )
{
	return list_buffer[list->offset + pos];
}

static int LIB_FirstList( void )
{
	return file_num;
}

static int LIB_IsList( int list )
{
	if ( list >= file_num )
		return 1;
	else
		return 0;
}

static char list_type[6][10] = {
	"Root",
	"Artist",
	"Album",
	"Title",
	"Playlist",
	"Genre",
};

static void lib_dump_files( void )
{
	int i;
	char path[256] = "";

	printf("\nLibrary Files:\n\n");
	for ( i = 0; i < file_num; i++ ) {
		printf("#%d", i );
		(void ) path_to_string( path, &files[i] );
		printf("\tPath:    %s\n", path );
		printf("\tFilename %s\n", get_string( files[i].name ) );
		if ( files[i].artist != -1 )
			printf("\tArtist:  %s\n", get_string( files[i].artist ) );
		if ( files[i].album != -1 )
			printf("\tAlbum:   %s\n", get_string( files[i].album ) );
		if ( files[i].title != -1 )
			printf("\tTitle:   %s\n", get_string( files[i].title ) );
		if ( files[i].track != 0 )
			printf("\tTrack:   #%d\n", files[i].track );

		printf("\tYear:    %d\n", files[i].year );
		printf("\tGenre:   #%d\n", files[i].genre );
		printf("\tType:    %d\n", files[i].type );
		printf("\tFlags:   %d\n", files[i].flags );
```

```
			printf("\n");
		}
}

static void tab( int len )
{
		int i;
		for( i = 0; i < len; i++ )
			printf("    ");
}

static void lib_dump_list( int list, int level );

static void lib_dump_list( int list, int level )
{
		struct list_str *l;
		int i;
		int item;

		l = LIB_GetList( list );

		tab( level );
		printf("LIST %s TYPE %s", get_string( l->name ), list_type[l->type] );
		if ( list == search_list )
			printf(" = SEARCH_LIST" );
		printf("\n");

		for ( i = 0; i < l->length; i++) {
			item = LIB_GetListItem( l, i );

			if ( LIB_IsList( item ) ) {
				lib_dump_list( item, level + 1 );
			} else {
				tab( level + 1 );

				printf("FILE ");
				if ( LIB_GetFile( item )->title != -1 )
					printf("#%02d Title: %s \n", LIB_GetFile( item )->track, get_string(
LIB_GetFile( item )->title ) );
				else
					printf("#%02d Name:  %s \n", LIB_GetFile( item )->track, get_string(
LIB_GetFile( item )->name ) );
			}
		}
}

static void lib_dump_lists( void )
{
		printf("\nLibrary Lists:\n\n");
		lib_dump_list( LIB_FirstList(), 0 );
}

static void lib_load( char* name )
{
		FILE *lib;
		int size;

		struct hdr_str *head;

		printf("\nLoading Library:\n\n");
		lib = fopen(  name, "r" );
		if( !lib ) {
```

```
		printf("file not found\n");
		exit(1);
	}
	size = fread( lib_buf, 1, LIB_BUF_MAX, lib );
	fclose( lib );
	printf("library file size %d \n", size );

	printf("\nLibrary Header:\n\n");
	head = (struct hdr_str *) lib_buf;
	printf("magic          %c%c%c%c \n", head->magic[0], head->magic[1], head->magic[2], head-
>magic[3] );
	printf("version        %08X \n", head->version );

	printf("files          %d\n", head->num_files);
	printf("lists          %d\n", head->num_lists);
	printf("ofs_files      %d\n", head->offset_files);
	printf("ofs_lists      %d\n", head->offset_list);
	printf("ofs_lists_entries  %d\n", head->offset_list_entries);
	printf("ofs_paths      %d\n", head->offset_paths);
	printf("ofs_string     %d\n", head->offset_strings);
	printf("ofs_private_data   %d\n", head->offset_private_data);

	// get number of files and lists
	file_num   = head->num_files;
	list_num   = head->num_lists;

	// setup all pointers
	files    = ( struct file_str* ) (lib_buf + head->offset_files);
	lists    = ( struct list_str* ) (lib_buf + head->offset_list);
	list_buffer = (unsigned short int*) (lib_buf + head->offset_list_entries );
	paths    = (char*) (lib_buf + head->offset_paths);
	strings  = (char*) (lib_buf + head->offset_strings);

	search_list = head->search_list;
	printf("search list     %d\n", search_list);
}

int main( int argc, char *argv[] )
{
	if (argc != 2) {
		printf("Usage: libdump filename\n\n" );
		exit(1);
	}

	lib_load( argv[1] );
	lib_dump_files();
	lib_dump_lists();
}
```