+$#**Contents**  
  
         Overview  
         Function List  
         DLLVersion  
         InitMaxSave  
         FreeMaxSave  
         LoadSave  
         NumberOfFiles  
         AddFileToSave  
         DeleteFileInSave  
         DeleteFileInSaveByName  
         SaveMaxFile  
         SetRootDir  
         FileExists  
         FileExistsInSavePos  
         ReplaceFileInSave  
         AddDataAsFile  
         FileDetails  
         CopyFileToBuffer  
         ExtractAFile  
         ExtractAFileAs  
         GetFileSize  
         AsciiToSJis  
         SJisToAscii  
         PS2 File System Limitations  
         Syntax Guide  
         .max limitations  
         License and credits  
         Changelog  
         Delphi Implementation  
         GetRootDir  
         Constants  
         Example code

+$#**Overview**  
  
ARMaxDLL is a fast and easy way to add support for AR Max saves to your program.  
  
The exported functions allow you to load and save .max save files as well as delete, add, replace and extract individual files. Helper functions for manipulating the browser text in icon.sys files have also been included.  
  
For a full list of functions see the function list.  
  
It is advisable to read the PS2 File System Limitations, .max limitations and the syntax guide before using the DLL.

+$#**Function List**  
  
DLLVersion  
InitMaxSave  
FreeMaxSave  
LoadSave  
NumberOfFiles  
AddFileToSave  
DeleteFileInSave  
DeleteFileInSaveByName  
SaveMaxFile  
SetRootDir  
GetRootDir  
FileExists  
FileExistsInSavePos  
ReplaceFileInSave  
AddDataAsFile  
FileDetails  
CopyFileToBuffer  
ExtractAFile  
ExtractAFileAs  
GetFileSize  
AsciiToSJis  
SJisToAscii

+$#**DLLVersion**  
  
**Overview**  
Returns the internal version of the DLL. Different revisions may use different parameters for functions, it is advisable to check the version number of the DLL before using any functions.  
  
**Declaration**  
function DLLVersion : integer; stdCall;  
  
Returns a 32 bit integer.  
  
**Usage**  
The DLL does not need to be initiated before use.  
  
*Example*  
  
version := DLLVersion;

+$#**InitMaxSave**  
  
**Overview**  
Initialises the internal data structures used when accessing .max saves. Required before using any other functions.  
  
**Declaration**  
function InitMaxSave : integer; stdCall;  
  
Returns a 32 bit integer.  
  
The result of a successful initialisation is 0.  
If the DLL is already initialised the result will be $F8 (constant ALREADY\_INITIALISED)  
  
**Usage**  
The DLL must not be initiated before use.  
  
*Example*  
  
theResult := InitMaxSave;

+$#**FreeMaxSave**  
  
**Overview**  
Releases the internal data structures used by the DLL. Must be the last function called when no longer working with the DLL.  
  
**Declaration**  
function FreeMaxSave: integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is already uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
*Example*  
  
theResult := FreeMaxSave;

+$#**LoadSave**  
  
**Overview**  
Loads an existing .max save into memory and allows direct access to the files contained.  
  
**Declaration**  
function LoadSave(fileName: PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
*Example*  
  
theResult := LoadSave(pathToFile);

+$#**NumberOfFiles**  
  
**Overview**  
Returns the amount of files within the .max save in memory.  
  
**Declaration**  
function NumberOfFiles : integer; stdCall;  
  
Returns a 32 bit integer.  
  
A successful result is a number greater than 0 but below 249.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
*Example*  
  
theResult := NumberOfFiles;

+$#**AddFileToSave**  
  
**Overview**  
Adds a file to the .max save in memory  
  
**Declaration**  
function AddFileToSave(fileName : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED).  
If there is already a file with the same filename the result will be $F7 (constant FILE\_ALREADY\_EXISTS).  
  
**Usage**  
The DLL must be initiated before use.  
  
*Example*  
  
theResult := AddFileToSave(pathtoFile);

+$#**DeleteFileInSave**  
  
**Overview**  
Deletes a file from the .max save in memory  
  
**Declaration**  
Function DeleteFileInSave(itemNum : integer): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The paramenter passed is the number of the file to delete from the internal list. Use a combination of NumberOfFiles and FileDetails to obtain a list of files.  
Note: The number passed has a direct relationship to the file listing, to delete the first file in the list pass 1 as the parameter. To delete the last save out of a list of 8 pass 8 as the parameter.  
  
*Example*  
  
theResult := DeleteFileInSave(1);

+$#**DeleteFileInSaveByName**  
  
**Overview**  
Deletes a file from the .max save in memory by comparing the filename passed to those in the internal memory.  
  
**Declaration**  
function DeleteFileInSaveByName(fileName : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The paramenter passed is the file name of the file to delete from the internal list. Use a combination of NumberOfFiles and FileDetails to obtain a list of files and their names.  
Note: The file name passed is **case sensitive**. Passing Icon.sys **will not** delete icon.sys from the file listing.  
  
*Example*  
  
theResult := DeleteFileInSaveByName(nameOfFile);

+$#**SaveMaxFile**  
  
**Overview**  
Creates a .max file with containing all the files in memory with the name and location specified.  
  
**Declaration**  
function SaveMaxFile(fileName : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The paramenter passed is the file name and location of where to save the .max file.  
Note: If no Root Dir name has been set (either by loading an existing .max file or by setting it using SetRootDir) the default text 'New Directory' will be used.  
  
*Example*  
  
theResult := SaveMaxFile(PathAndFileName);

+$#**SetRootDir**  
  
**Overview**  
Sets the Root Directory of the .max save. This is the directory the files will be extracted to on the PS2 Memory card.  
  
**Declaration**  
function SetRootDir(name : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The paramenter passed is the name of the parent directory for all the files on the memory card.  
Note: If no Root Directory name has been set (either by loading an existing .max file or by setting it using SetRootDir) when saving a .max file the default text 'New Directory' will be used.  
  
Text over 31 printable characters in length will be trucated.  
  
*Example*  
  
theResult := SetRootDir(nameOfDirectory);

+$#**FileExists**  
  
**Overview**  
Does a case sensative search for a filename in the files in memory.  
  
**Declaration**  
function FileExistsInSave(name : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 1, an usuccesfull result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The paramenter passed is the name of file to search for.  
Note: The file name passed is **case sensitive**. Passing Icon.sys will return false (0) if the name is icon.sys.  
  
*Example*  
  
theResult := FileExistsInSave(nameOfFile);

+$#**FileExistsInSavePos**  
  
**Overview**  
Does a case sensitive search for a filename in the files in memory and returns it's position in the list.  
  
**Declaration**  
function FileExistsInSavePos(name : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is a value greater than 0, an usuccessful result (filename not found) is -1.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The paramenter passed is the name of file to search for.  
Note: The file name passed is **case sensitive**. Passing Icon.sys will return false (-1) if the name is icon.sys.  
  
*Example*  
  
theResult := FileExistsInSavePos(nameOfFile);

+$#**ReplaceFileInSave**  
  
**Overview**  
Replaces an existing file with the data of another file. The existing filename passed is case sensitive. The filename of the data to replace with does not need to match the existing filename as this name will not be used in the .max save.  
  
**Declaration**  
function ReplaceFileInSave(existingFileName : PChar; newFile : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The first paramenter passed is the name of file whose data will be replaced, the second parameter is the location and filename of the file whose data will replace the originals.  
Note: The file name passed as the first parameter is **case sensitive**. Passing Icon.sys not work if the name is icon.sys.  
  
*Example*  
  
theResult := ReplaceFileInSave(nameOfFile, locationAndNameOfAnotherFile);

+$#**AddDataAsFile**  
  
**Overview**  
Adds data in memory as a file to the internal .max file listing.  
  
**Declaration**  
function AddDataAsFile(buffer :PChar; bufferSize : integer; name : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED).  
If the filename passed already exists the result will be $F7 (constant FILE\_ALREADY\_EXISTS).  
  
**Usage**  
The DLL must be initiated before use.  
  
The first parameter is the data to be added, this is a pointer to an array of data (PCHAR).  
The second parameter is the size of the data.  
The third parameter is the name the data should be stored as.  
  
*Example*  
  
**var**  
         buffer : PChar;  
         theResult : integer;  
  
**begin**  
         getmem(buffer, 32);  
         fillchar(buffer, 32, $01);  
         theResult := AddDataAsFile(buffer, 32, PChar('a file.bin');  
         freemem(buffer);  
**end;**

+$#**FileDetails**  
  
**Overview**  
Gets the file name and size of a file stored in memory  
  
**Declaration**  
function FileDetails(itemNum: integer; name: PChar; nameLength: integer; var fileSize: integer): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED).  
Passing a number greater than the amount of files will result in $F6 (constant NO\_ITEM\_NUM)  
  
**Usage**  
The DLL must be initiated before use.  
  
The *itemNum* parameter is the number of the file in the listing. To find out how many files there are in memory use NumberOfFiles.  
The number passed has a direct relationship to the file listing, to get the details of the first file in the list pass 1 as the parameter.  
The second parameter *name* should be a PCHAR (pointer to an array of char) or 32 bytes with memory already allocated.  
The third parameter *nameLength* is the size of the *name* parameter.  
The fourth parameter *fileSize* is a integer for storing the file size of the file. This should be passed by **var** in Delphi, by Ref in C.  
  
If for any reason the fileSize paramater is returning false results (a size of 0 etc) then use GetFileSize to get a files size.  
  
*Example*  
  
**var**  
         buffer : PChar;  
         theResult, fileSize : integer;  
  
**begin**  
         getmem(buffer, 32);  
         theResult := FileDetails(1, buffer, 32, fileSize);  
         *//buffer now contains the file name of the 1st file.*  
         freemem(buffer);  
**end;**

+$#**CopyFileToBuffer**  
  
**Overview**  
Copys a file from memory into a supplied buffer.  
  
**Declaration**  
function CopyFileToBuffer(itemNum : integer; buffer : PChar; bufferSize : integer): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED).  
If for some reason this fails the result will be $F5 (constant EXTRACT\_FAILED).  
  
**Usage**  
The DLL must be initiated before use.  
  
The *itemNum* parameter is the number of the file in the listing. To find out how many files there in memory use NumberOfFiles.  
The *buffer* parameter is a buffer to contain the file data, this should already have a suitable amount of memory allocated.  
The *bufferSize* parameter is the size of the buffer passed.  
  
To find out a files size use FileDetails or GetFileSize.  
  
*Example*  
  
**var**  
         buffer : PChar;  
         theResult, fileSize : integer;  
  
**begin**  
         fileSize := GetFileSize(1);  
         getmem(buffer, fileSize);  
         theResult := CopyFileToBuffer(1, buffer, fileSize);  
         *//buffer now contains the data of the 1st file.*  
         freemem(buffer);  
**end;**

+$#**ExtractAFile**  
  
**Overview**  
Extracts a file to a location supplied.  
  
**Declaration**  
function ExtractAFile(itemNum : integer; location : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED).  
If for some reason this fails the result will be $F5 (constant EXTRACT\_FAILED).  
  
**Usage**  
The DLL must be initiated before use.  
  
The *itemNum* parameter is the number of the file in the listing. To find out how many files there are in memory use NumberOfFiles.  
The number passed has a direct relationship to the file listing, to get the details of the first file in the list pass 1 as the parameter.  
The *location* parameter is the location to save the file and **must included** a trailing backslash.  
  
The file will be extracted with it's current file name to the location supplied.  
  
Note: PS2 file names can use characters **not** supported by Windows. It is advisable to use ExtractAFileAs and specify 'Windows Safe' filenames.  
  
*Example*  
  
theResult := ExtractAFile(1, locationToSaveTo);

+$#**ExtractAFileAs**  
  
**Overview**  
Extracts a file using the location and file name supplied.  
  
**Declaration**  
function ExtractAFileAs(itemNum : integer; fileName : PChar): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED).  
If for some reason this fails the result will be $F5 (constant EXTRACT\_FAILED).  
  
**Usage**  
The DLL must be initiated before use.  
  
The *itemNum* parameter is the number of the file in the listing. To find out how many files there are in memory use NumberOfFiles.  
The number passed has a direct relationship to the file listing, to get the details of the first file in the list pass 1 as the parameter.  
The *fileName* parameter is the location and file name to save the file as.  
  
*Example*  
  
theResult := ExtractAFile(1, locationAndFileName);

+$#**GetFileSize**  
  
**Overview**  
Returns the size of a file in memory  
  
**Declaration**  
function GetFileSize(itemNum : integer): integer; stdCall;  
  
Returns a 32 bit integer.  
  
A successful result is the size of a file, files can be 0 size.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The *itemNum* parameter is the number of the file in the listing. To find out how many files there are in memory use NumberOfFiles.  
The number passed has a direct relationship to the file listing, to get the details of the first file in the list pass 1 as the parameter.  
  
*Example*  
  
fileSize := GetFileSize(1);

+$#**AsciiToSJis**  
  
**Overview**  
A helper function for handling the browser text in the icon.sys file. Takes a letter/character of text (CHAR) and returns a word/shortInt value to use directly in the file  
  
**Declaration**  
function AsciiToSJis(input : char): shortInt; stdCall;  
  
Returns a word/ShortInt value  
  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
Pass a letter or char (CHAR), the result is the SJIS equivilent ready for immediate use in the icon.sys file (no byteswap needed).  
The PS2 file system does not support the following characters: '\*'(0x2a), '/'(0x2f), and '?'(0x3f), these are returned as a space.  
Any unsupported or unknown character is returned as a space.  
  
*Example*  
  
theResult := AsciiToSJis('A');

+$#**SJisToAscii**  
  
**Overview**  
A helper function for handling the browser text in the icon.sys file. Takes the 2 bytes used per character in the icon.sys file and returns the English equivilent.  
  
**Declaration**  
function SJisToAscii(input : shortInt): char; stdCall;  
  
Returns a Char value  
  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
Pass the 2 bytes used in per letter in the icon.sys file to get the English version.  
This function can compensate for incorrect space characters generated by PS2 Save Builder and mcIconSysGen and translates them to a space.  
  
Unknown entries are returned as '?'. This is a non-supported character and if written back to the file will be translated into a space.  
  
*Example*  
  
**var**  
         a : integer;  
         buffer : string;  
  
**begin**  
         buffer := '';  
         for a := 0 to 33 do begin  
        buffer := buffer + SJisToAscii(iconFile.titleName[a]);  
        end;

+$#**PS2 File System Limitations**  
  
**PS2 File System Limitations**  
  
 File names must be no more than 32 bytes long (31 characters and a trailing 0x00)  
 File names cannot contain the following characters: '\*'(0x2a), '/'(0x2f), and '?'(0x3f)  
 The valid ascii range for use is 0x20 - 0x7e  
 All file names are case sensitive  
 The maximum amount of files you can have in a directory is 18\*  
  
\*A valid PS2 save must have an icon.sys file and at least one icon. SCEE guidelines also request a file named the same as the directory. This means a directory can contain a maximum of 15 user data files but usually a maximum of 16 can be used by omitting the file named the same as the directory.

+$#**Syntax Guide**  
  
**Syntax used in this document**  
  
The syntax used in this document is based on Delphi AKA Object Pascal. It is easy to read and should be quite clear on how each function works.  
  
Hex values are often represented like $5A as opposed to the C formatting of 0x5A.  
PChar is a pointer to an array of char.

+$#K**.max limitations**  
  
**.max limitations**  
  
The AR Max ave format has no provsion for recording file attributes, files are restored using a pre-determined generic set of attributes by the AR Max software.  
It is unlikely that the AR Max save format supports PSOne saves or PocketStation files as these require a specific file system attribute to be set. Further experimentation is required to confirm this.

+$#**License and credits**  
  
**Licence**  
You are free to use this DLL in any free product or program. For usage with commercial products please contact the author.  
  
**Author**  
gothi - gothi@ps2savetools.com  
 http://www.ps2savetools.com  
  
**Thanks**  
Angie - patience and looks of an Angel.  
Ross Ridge - for his sterling work on  mymc

+$#K**Changelog**  
  
**Version 2** - 29/07/07  
Renamed project to ARMaxDLL to avoid naming clash with existing projects  
Updated some internal routines for optimised string handling.  
Added GetRootDir function  
Updated help file.  
  
  
**Version 1** - 22/07/07  
First release.

+$#**Delphi Implementation**  
  
function InitMaxSave: integer; stdcall; external 'ARMaxDLL.dll';  
function FreeMaxSave: integer; stdcall; external 'ARMaxDLL.dll';  
function LoadSave(fileName : PChar): integer; stdcall; external 'ARMaxDLL.dll';  
function NumberOfFiles: integer; stdcall; external 'ARMaxDLL.dll';  
function FileDetails(itemNum: integer; name: PChar; nameLength: integer; var fileSize: integer): integer; stdCall; external 'ARMaxDLL.dll';  
function AddFileToSave(fileName : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
Function DeleteFileInSave(itemNum : integer): integer; stdCall; external 'ARMaxDLL.dll';  
function DeleteFileInSaveByName(fileName : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function SaveMaxFile(fileName : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function SetRootDir(name : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function FileExistsInSave(name : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function FileExistsInSavePos(name : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function ReplaceFileInSave(existingFileName : PChar; newFile : Pchar): integer; stdCall; external 'ARMaxDLL.dll';  
function AddDataAsFile(buffer :PChar; bufferSize : integer; name : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function CopyFileToBuffer(itemNum : integer; buffer : PChar; bufferSize : integer): integer; stdCall; external 'ARMaxDLL.dll';  
function ExtractAFile(itemNum : integer; location : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function ExtractAFileAs(itemNum : integer; fileName : PChar): integer; stdCall; external 'ARMaxDLL.dll';  
function GetFileSize(itemNum : integer): integer; stdCall; external 'ARMaxDLL.dll';  
function AsciiToSJis(input : char): shortInt; stdCall; external 'ARMaxDLL.dll';  
function SJisToAscii(input : shortInt): char; stdCall; external 'ARMaxDLL.dll';  
function GetRootDir(buffer : PChar; buffersize : integer): integer; stdCall; external 'ARMaxDLL.dll';

+$#**GetRootDir**  
  
**Overview**  
Gets the Root Directory of the .max save. This is the directory the files will be extracted to on the PS2 Memory card.  
  
**Declaration**  
function GetRootDir(buffer : PChar; buffersize : integer): integer; stdCall;  
  
Returns a 32 bit integer.  
  
Successful result is 0.  
If the DLL is uninitialised the result will be $F9 (constant NOT\_INITIALISED)  
  
**Usage**  
The DLL must be initiated before use.  
  
The buffer passed should be a pointer to 32 bytes long (31 chars + trailing 0x00) array of char.  
  
*Example*  
  
theResult := GetRootDir(nameOfDirectory, 32);

+$#**Constants**  
  
**Constants**  
  
The DLL uses the following internal Constants:  
  
NOT\_INITIALISED = $F9;  
ALREADY\_INITIALISED = $F8;  
FILE\_ALREADY\_EXISTS = $F7;  
NO\_ITEM\_NUM = $F6;  
EXTRACT\_FAILED = $F5;  
BUFFER\_SIZE\_ERROR = $F4;

+$#K**Example code**  
  
**Example Delphi code**  
  
**Opening and listing contents of an existing .max save:**  
  
procedure TForm1.Button1Click(Sender: TObject);  
var  
theResult, a, fileSize : integer;  
aName : PChar;  
begin  
if OpenDialog1.Execute then begin  
InitMaxSave;  
LoadSave(PChar(OpenDialog1.FileName));  
theResult := NumberOfFiles;  
memo1.lines.add('Number of Files: ' + intToStr(theResult));  
getMem(aName, 32);  
for a := 1 to theResult do begin  
theResult := fileDetails(a, aName, 32, fileSize);  
memo1.Lines.Add(aName + ' ' + intToStr(fileSize));  
end;  
GetRootDir(aName, 32);  
memo1.Lines.Add('Root Dir: ' + aName);  
FreeMem(aName);  
FreeMaxSave;  
end;  
end;  
  
**Creating a new .max save, adding a file and saving:**  
  
procedure TForm1.AddFileandSaveNewButton(Sender: TObject);  
begin  
if opendialog1.Execute then begin  
InitMaxSave;  
addFileToSave(PChar(openDialog1.FileName));  
setRootDir(PChar('My Files'));  
if SaveDialog1.Execute then begin  
SaveMaxFile(PChar(saveDialog1.FileName));  
end;  
FreeMaxSave;  
end;  
end;