

Soundkarten-Messung I : Steuerung mit der BASS.DLL

[Druckfassung](#)

von Michael Gaedtke

Letzte Änderungen vom 23. Juli 2006

- [1. Die BASS.DLL unter Delphi](#)
- [2. Soundkartensteuerung](#)
- [3. Initialisierung der Soundkarte - Create und Destroy](#)
- [4. Mixer-Steuerung](#)
- [5. Das Herzstück: Die Callback-Routine](#)
- [6. Pushing oder Pulling?](#)
- [7. Aufnahme starten und stoppen](#)
- [8. Zum Schluss: Ein einfacher StatusBar](#)
- [9. Downloads](#)

BASS ist eine Dynamic Link Library – abgekürzt DLL – die Methoden für die Einbindung der Soundkarte und für vielfältige Manipulationen von Sounddateien zur Verfügung stellt. Für das spezielle Interesse, die Soundkarte auch für Messzwecke nutzbar zu machen, ist entscheidend, dass die BASS-DLL auch Routinen zur Aufzeichnung über die Soundkarte umfasst. Sie steht zusammen mit diversem Zubehör unter <http://www.un4seen.com> zum Download bereit. Die BASS-Library läuft unter Windows und Mac OS X und bietet Programmentwicklern sehr effiziente und einfach anzuwendende Software-Routinen, die den Zugriff auf Samples, Streams im MP3, MP2, MP1, OGG, WAV, AIFF-Format sowie MOD (Music On Demand) in den verschiedensten Formaten ermöglicht. Dabei bildet die gesamte DLL ein Paket, das gerade einmal knapp unter 100 kByte groß ist.

Unter Windows setzt BASS DirectX ab Version 3 voraus und nutzt – soweit vorhanden – beschleunigte DirectSound und DirectSound3D Treiber. Zum System gehören neben der hier

eingesetzten Delphi-Anbindung (Datei BASS.pas) auch APIs für C/C++, Visual Basic und MASM-Programme, die zusammen mit ausführlichen Beispielen und einer umfangreichen Dokumentation geliefert werden. Der Einsatz der BASS-Library ist für nicht kommerzielle Anwendungen frei. Wenn die selbst programmierte Software kostenfrei weitergegeben wird und keinen anderen kommerziellen Hintergrund hat (z.B. in der Werbung), dann darf BASS kostenfrei eingesetzt werden. Andernfalls muss eine Lizenz angeschafft werden. Weitere Informationen zur Lizenzierung unter <http://www.un4seen.com>.

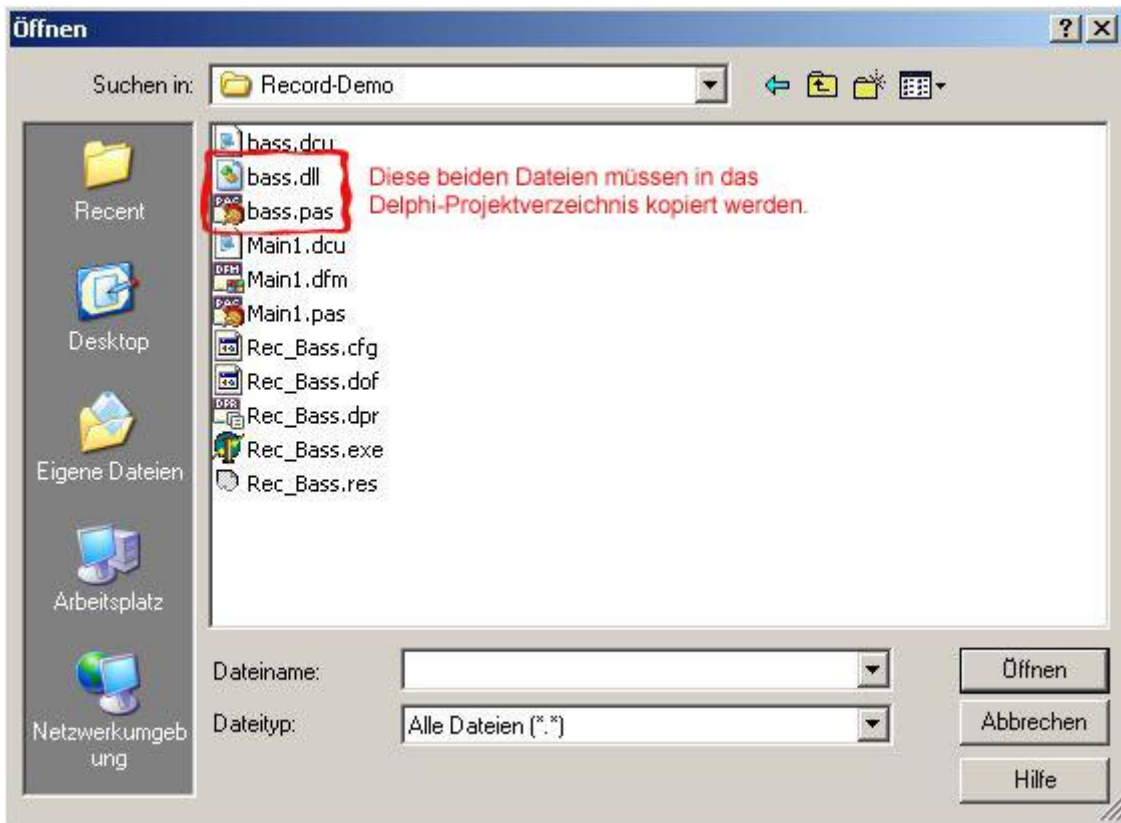
Eine Dynamic Link Library ist eine unter Microsoft Windows nutzbare Bibliothek mit Programmroutinen und gibt sich meist mit der Dateierweiterung *.dll zu erkennen. Windows selbst besteht zum großen Teil aus DLLs. Sie wurden ursprünglich eingeführt, um Speicherplatz auf der Festplatte zu sparen. Programmcode, der von mehr als einer Anwendung benötigt wird, braucht so im Prinzip nur einmal vorgehalten und auch in den Hauptspeicher geladen zu werden. Die mit dem Bibliothekskonzept einher gehende Modularität hat für Programmanbieter den zusätzlichen Vorteil, dass man mit kleineren Service-Packs auskommt. Allerdings kann es nach einiger Zeit dazu kommen, dass verschiedene Programme unterschiedliche Versionen einer DLL benötigen (DLL Hell). Solche Konflikte können umgangen werden, wenn die jeweils zugehörige DLL immer in das gleiche Verzeichnis wie das aufrufende Programm geladen werden. Dadurch geht zwar der Vorteil der Speicherplatzersparnis zumindest partiell wieder verloren. Bei den heute zur Verfügung stehenden Festplattenkapazitäten spielt das aber keine entscheidende Rolle mehr. Im nachfolgenden Beispiel werden wir diese direkte Zuordnung von EXE und DLL in einem Verzeichnis praktizieren. Eine Alternative wäre, die DLL in das Windows-Systemverzeichnis zu kopieren. Ein weiterer Vorteil von DLLs besteht darin, dass ihre Methoden unabhängig von der Sprache, in der sie programmiert sind, auch von anderen Programmiersprachen aus genutzt werden können. So kann man aus einer Delphi-Anwendung auf eine C-DLL zugreifen und umgekehrt.

Die BASS-Library ist von [Ian Luck](#) programmiert worden, der mir freundlicherweise auch einige Fragen, die beim Experimentieren für dieses Tutorial aufgetaucht waren, sehr schnell beantwortet hat. Vielen Dank dafür! Wer noch nicht mit Dynamic Link Libraries gearbeitet hat und sich näher für die Hintergründe interessiert, sollte sich den Lexikon-Eintrag zu [DLLs](#) in der Wikipedia oder das [DLL-Tutorial](#) von Martin Strohal bei dsdt.info ansehen.

[Top](#)

1. Die BASS.DLL unter Delphi

Voraussetzung für die Nutzung der BASS-Routinen ist, dass die DLL in die eigene Anwendung eingebunden wird. Für ein neues Projekt legt man dazu ein eigenes Verzeichnis an, in das alle zum Projekt gehörenden Dateien kommen. In dieses Verzeichnis kopiert man auch die BASS.dll und die zugehörige Unit BASS.pas, die die DLL für Delphi portiert. Später muss die DLL-Datei auch mit der fertigen EXE in einem Verzeichnis stehen.



In den Interface-Teil des neuen Programms nimmt man in die uses-Klausel die BASS-Unit auf:

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, ComCtrls,
```

```
BASS;
```

In der Unit BASS.pas sind die Funktionsaufrufe der externen DLL zusammengefasst. Das könnte man zwar auch innerhalb des eigenen Programms erledigen, würde den Quelltest wegen der Funktionsvielfalt aber völlig unübersichtlich machen. Zur Verdeutlichung, wie die Funktionseinbindung der DLL funktioniert, folgt ein kurzer Ausschnitt aus der BASS.pas-Unit:

```
// Functions  
const  
    bassdll = 'bass.dll';  
  
function BASS_SetConfig(option, value: DWORD): DWORD; stdcall;  
    external bassdll;  
function BASS_GetConfig(option: DWORD): DWORD; stdcall;  
    external bassdll;  
function BASS_GetVersion: DWORD; stdcall; external bassdll;  
function BASS_ErrorGetCode: Integer; stdcall; external bassdll;  
function BASS_Init(device: Integer; freq, flags: DWORD; win: HWND;  
    clsid: PGUID): BOOL; stdcall; external bassdll;
```

```

function BASS_SetDevice(device: DWORD): BOOL; stdcall;
    external bas.dll;
function BASS_GetDevice: DWORD; stdcall;
    external bas.dll;
function BASS_Free: BOOL; stdcall;
    external bas.dll;
function BASS_GetDSoundObject(obj: DWORD): Pointer; stdcall;
    external bas.dll;
function BASS_GetInfo(var info: BASS_INFO): BOOL; stdcall;
    external bas.dll;
function BASS_Update: BOOL; stdcall; external bas.dll;
function BASS_GetCPU: FLOAT; stdcall; external bas.dll;
function BASS_Start: BOOL; stdcall; external bas.dll;
function BASS_Stop: BOOL; stdcall; external bas.dll;
function BASS_Pause: BOOL; stdcall; external bas.dll;
function BASS_SetVolume(volume: DWORD): BOOL; stdcall;
    external bas.dll;
function BASS_GetVolume: Integer; stdcall; external bas.dll;

```

usw.

Zum Einbinden einer DLL-Routine in eigene Programme muss lediglich das Schlüsselwort `external` im Anschluss an den Funktions- oder Prozedurkopf aufgerufen werden. Mit dieser Methode wird die DLL unmittelbar beim Start des Programms in die Anwendung eingebunden. Es ist auch möglich, DLL-Routinen dynamisch zur Programmlaufzeit einzubinden, wenn sie tatsächlich gebraucht werden, aber das ist erheblich komplizierter und lohnt sich allenfalls bei sehr großen Programmkasketen und/oder DLLs mit hunderten von Funktionen. Der Befehl `stdcall` sorgt dafür, dass die Parameter in der für Windows erforderlich Art und Weise übergeben werden. Außerdem werden in der `BASS.pas`-Unit zahlreiche Konstanten und Typen deklariert; auch dazu folgt ein Beispiel:

```

type
    DWORD = Cardinal;
    BOOL = LongBool;
    FLOAT = Single;
    QWORD = Int64;

    HMUSIC = DWORD;           // MOD music handle
    HSAMPLE = DWORD;         // sample handle
    HCHANNEL = DWORD;       // playing sample's channel handle
    HSTREAM = DWORD;        // sample stream handle
    HRECORD = DWORD;        // recording handle
    HSYNC = DWORD;          // synchronizer handle
    HDSP = DWORD;           // DSP handle
    HFX = DWORD;            // DX8 effect handle
    HPLUGIN = DWORD;        // Plugin handle

```

usw.

Die Unit `BASS.pas` ist dem BASS-Downloadpaket als Quelltext beigelegt, so dass man sich die Definitionen und Funktionsaufrufe ansehen kann. Zur Einführung in die Hintergründe der BASS-Library soll das genügen.

[Top](#)

2. Soundkartensteuerung

Auch in unserem Beispielprogramm wird im Anschluss an die Einbindung der Unit zunächst eine Reihe von Konstanten vereinbart. Es empfiehlt sich, im Quelltext nicht mit "magic numbers" zu arbeiten, sondern alle Parameter der Soundkartensteuerung wie Abtastrate, Kanalzahl und Auflösung nur einmal an zentraler Stelle zu definieren, damit spätere Änderungen ohne aufwändige Suchaktion vorgenommen werden können:

```
const
  cDefaultDevice = -1;      // Default Device Identifier
  cSampleRate = 44100;     // PCM-Audio
  cNumChannels = 2;        // Stereo
  cRecordingTime = 200;    // ms (10 - 500 ms / Default 100 ms)
  cBufferBlocks = 44100;   // Länge des Puffers für Audiosamples
  c16BitAudio = 0;        // Flag für 16 Bit Audio
  cMaxAudio = 32768;       // maximaler Pegel bei 16 Bit
  cDefaultUser = 0;       // UserIDentifier (not used)
  cDirectXPointer = nil;   // Pointer für DirectX Class Identifier
  Le = 0;                  // Left Channel
  Ri = 1;                  // Right Channel
```

Die von der Soundkarte aufgenommenen Samplewerte werden für die Weiterverarbeitung in einen lokalen Puffer geschrieben. Das Format des Audiopuffers muss am den Soundkarten- und Wave-Datei-Standard angepasst sein. Da wir mit 16 Bit Audio (CD-Qualität) arbeiten wollen, muss für jedes Sample pro Kanal eine 2 Bytes Integervariable mit Vorzeichen bereitgestellt werden. Das entspricht in Delphi dem Typ `SmallInt`. Für ein Stereosample werden daher insgesamt vier Bytes benötigt. In der üblichen Reihenfolge der Sampledaten kommt zuerst das Sample des linken Kanals gefolgt vom zugehörigen Sample des rechten Kanals. Das wird durch ein zweidimensionales Array verwirklicht:

```
type
  TAudioSample = SmallInt; // 16 Bit mit Vorzeichen
  TStereoSample = array[Le..Ri] of TAudioSample;
  TStereoBuffer = array[0..cBufferBlocks] of TStereoSample;
```

Mit einer Puffervariablen dieses Typs kann im Programm gezielt und komfortabel auf die Links- und Rechts-Information jedes einzelnen Samples zugegriffen werden:

```
var
  LocalBuffer: TStereoBuffer;
  EinSample: TAudioSample; // 2 Bytes SmallInt
  i: cardinal;

begin
  CopyMemory(@LocalBuffer, Buffer, BufLength);
  for i := 0 to 100 do begin
    EinSample := LocalBuffer[i, Le];
  end; // for
end;
```

Das ist nur ein Beispiel. Hier wird zweckfrei nacheinander auf 101 Samples des linken Kanals zugegriffen. An diese Typvereinbarung schließt sich die Typ- und Variablendeklaration des Projekts an.

```
type
  InputChannelsComboBox: TComboBox;
  Label1: TLabel;
  StatusBar1: TStatusBar;
  RecordButton: TButton;
  RecordPegelBar: TTrackBar;
  StopButton: TButton;
  ProgressBar1: TProgressBar;
  Label2: TLabel;
  Label3: TLabel;
  procedure FormCreate(Sender: TObject);
  procedure InputChannelsComboBoxChange(Sender: TObject);
  procedure RecordPegelBarChange(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
  procedure RecordButtonClick(Sender: TObject);
  procedure StopButtonClick(Sender: TObject);
private
  procedure UpdateStatusInfo;
end;

var
  Form1: TForm1;
  RecChannel: HRecord;
```

Die globale Variable RecChannel nimmt ein Handle auf, den die BASS.DLL zur Identifizierung einen laufenden Recording-Prozesses benutzt.

[Top](#)

3. Initialisierung der Soundkarte - Create und Destroy

Damit ist der Interface-Teil der Applikation bereits abgeschlossen und wir kommen zur Implementation: In der FormCreate-Prozedur wird beim Programmstart zunächst geprüft, ob die korrekte Version von BASS geladen werden kann, andernfalls wird eine Fehlermeldung ausgegeben und die Ausführung abgebrochen:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i : byte;
  Par : Integer;
  ChannelName : PChar;
begin
  if (HiWord(BASS_GetVersion) <> BassVersion)
  then begin
    MessageBox(0, 'Falsche Version der BASS.dll geladen!',
      nil ,MB_IconError);
```

```

    Halt;
end; // then

```

Die Versionskennung der DLL wird im HiWord-Teil der Funktionsantwort von BASS_GetVersion zurück geliefert. Dann wird mit den beiden Funktionen Bass_Init und BASS_RecordInit geprüft, ob die Soundkarte für die Aufnahme angesprochen werden kann:

```

if (not BASS_RecordInit(cDefaultDevice))
or (not BASS_Init(cDefaultDevice,
                  cSampleRate,      // Samplerate
                  c16BitAudio,     // Flags; 0 = 16 Bit Audio
                  Handle,          // Window-Applikation-Handle
                  cDirectXPointer)) // Pointer für DirectX Class
then begin
    BASS_RecordFree;
    BASS_Free;
    MessageBox(0, 'Default-Aufnahmegerät kann
                  nicht gestartet werden!',
               nil, MB_IconError);

    Halt;
end; // then

```

BASS_RecordInit müssen eine Reihe von Parametern übergeben werden, die festlegen, welche Soundkarte (hier: Default) mit welcher SampleRate (44100) und welcher Auflösung (16 Bit) aktiviert werden soll. Wenn die Initialisierung erfolgreich war, wird jetzt ermittelt, welche Eingangskanäle der Soundkarte zur Verfügung stehen. Dazu sieht die DLL die Funktion BASS_RecordGetInputName vor.

```

i := 0;
ChannelName := BASS_RecordGetInputName(i);
while ChannelName <> nil do
    begin
        InputChannelsComboBox.Items.Add(StrPas(ChannelName));
        if (BASS_RecordGetInput(i) and BASS_Input_off) = 0
            then InputChannelsComboBox.ItemIndex := i;
        Inc(i);
        ChannelName := BASS_RecordGetInputName(i);
    end; // while

    InputChannelsComboBoxChange(Self);

```

BASS_RecordGetInputName liefert im Erfolgsfall einen Zeiger auf eine Beschreibung des Recording-Kanals zurück, sonst wird nil zurückgegeben. Die Funktion wird in einer While-Schleife so lange mit hochgezähltem Index i aufgerufen, bis nil zurückgegeben wird. Mit der Funktion BASS_RecordGetInput können die aktuellen Einstellungen eines Eingangskanals ermittelt werden. Im Fehlerfall wird -1 zurückgegeben. Im Erfolgsfall wird im LoWord-Teil der eingestellte Lautstärkepegel als Prozentwert (0 - 100) zurückgegeben. Mit Hilfe des BASS_Input_off Flags wird geprüft, ob der Kanal abgeschaltet (disabled) ist. Mit der eigenen Prozedur InputChannelsComboBoxChange wird sichergestellt, dass der ausgewählte Eingangskanal aktiviert ist und die Statusanzeige des Programms aktualisiert.

Beim Schließen des Programms müssen die von der DLL belegten Speicherbereiche wieder frei gegeben werden. Dazu dienen die Funktionen BASS_RecordFree die alle Ressourcen der Recording-Einheit freigibt, und BASS_Free die alle Ressourcen der Eingangs-Einheit

einschließlich der Samples und Streams freigibt. Die Funktionen geben im Fehlerfall den booleschen Wert false zurück; der Rückgabewert braucht aber nicht zwingend ausgewertet zu werden. Mit BASS_Stop werden sicherheitshalber alle Ausgaben gestoppt.

```
procedure TForm1.FormDestroy(Sender: TObject);  
  begin  
    BASS_RecordFree;  
    BASS_Free;  
    BASS_Stop;  
  end; // procedure FormClose
```

[Top](#)

4. Mixer-Steuerung

Die BASS.dll stellt Routinen bereit, mit denen die Einstellungen des Windows-Mixers online verändert werden können. Damit können wir aus unserem Programm heraus den Eingangskanal auswählen und den gewünschten Aufnahmepegel einstellen. Für die Einstellung eines ausgewählten Eingangskanals gibt es die Funktion BASS_RecordSetInput, die bei erfolgreichem Aufruf den booleschen Wert true zurückgibt. Die Eingangskanäle werden anhand ihrer laufenden Nummer aufgerufen. In unserem Beispielprogramm werden die von der Soundkarte zur Verfügung gestellten Kanäle komfortabel mit einer ComboBox verwaltet. Um einen neu ausgewählten Eingangskanal zu aktivieren, werden zunächst alle vorhandenen Kanäle deaktiviert, indem wiederholt die Funktion BASS_RecordSetInput in einer While-Schleife mit dem Flag BASS_Input_off aufgerufen wird. Der in der ComboBox ausgewählte Kanal wird anschließend mit dem Flag BASS_Input_on aktiviert.

```
procedure TForm1.InputChannelsComboBoxChange(Sender: TObject);  
  var  
    i : byte;  
    r : boolean;  
  begin  
    r := true;  
    i := 0;  
    while r do begin  
      r := BASS_RecordSetInput(i, BASS_Input_off);  
      Inc(i);  
    end; // while  
    BASS_RecordSetInput(InputChannelsComboBox.ItemIndex,  
                        BASS_Input_On);
```

Der für diesen Kanal im Windows-Mixer eingestellte Aufnahmepegel soll auf den Schieberegler unseres Programms übertragen werden. Dazu wird seine Stellung mit der Funktion BASS_RecordGetInput abgefragt; die Position des Reglers wird im LoWord des Rückgabewerts als Prozentwert (0 - 100) zurückgegeben und auf die Positions-Eigenschaft des RecordPegelBar-Reglers übertragen. Schließlich wird der aktivierte Kanal im StatusBar angezeigt:

```
with InputChannelsComboBox do begin  
  RecordPegelBar.Position :=
```

```

    LoWord(BASS_RecordGetInput(ItemIndex));
end; // with

UpdateStatusInfo;
end; // procedure InputChannelComboBoxChange

```

Umgekehrt kann der Aufnahmepegel des Windows-Mixers auch aus unserem Programm heraus eingestellt werden. Dazu verwenden wir wieder die Funktion `BASS_RecordSetInput`, der ein Prozentwert zwischen 0 und 100 übergeben wird. Mit einer `or`-Verknüpfung wird im Beispiel direkt der Positions-Wert des `RecordPegelBar`-Schiebereglers auf das `LoWord` des `BASS_Input_Level` übertragen. Wenn man den Recording-Teil des Windows-Mixers parallel zum Programm öffnet, kann man ausprobieren, wie die Schieberegler "ferngelenkt" bewegt werden können.

```

procedure TForm1.RecordPegelBarChange(Sender: TObject);
begin
    with InputChannelsComboBox do begin
        BASS_RecordSetInput(ItemIndex, BASS_Input_Level
                               or RecordPegelBar.Position);
    end; // with
end; // procedure RecordPegelBarChange

```

[Top](#)

5. Das Herzstück: Die Callback-Routine

Windows-Programme funktionieren ereignisgesteuert. Die Soundkarte füllt bei der Aufnahme Speicherblöcke als Datenpuffer und sendet eine Nachricht aus, wenn der aktuelle Puffer gefüllt und abholbereit ist. Auf dieses Ereignis muss unser Programm reagieren, damit keine Daten verloren gehen. Dazu wird eine Callback-Routine benötigt, die Windows immer dann aufruft, wenn ein Soundkarten-Event auftritt. Eine solche Callback-Funktion muss nach einem festgelegten Muster aufgebaut sein, damit Windows die Parameter in der korrekten Reihenfolge übergeben kann. Damit die Parameterübergabe an das Betriebssystem funktioniert, muss die Aufrufkonvention `stdcall` gesetzt sein.

```

function RecordingCallback (Handle : HRecord;
                             Buffer : Pointer;
                             BufLength : DWord;
                             User : DWord) : boolean; stdcall;

var
    LocalBuffer: TStereoBuffer;
    Sum, Mean, Root : extended;
    RMSLevel : integer;
    Level : integer;
    LeftLevel : word;
    RightLevel : word;
    i : cardinal;
begin
    Form1.Label2.Caption := 'Buffer-Länge: ' + IntToStr(BufLength);

    CopyMemory(@LocalBuffer, Buffer, BufLength);

```

Diese Funktion ist das Herzstück der Aufnahme. Mit dem Handle identifiziert sich die Quelle der Daten; in unserem Fall ist das die Applikation. Die Variable Buffer übergibt einen Zeiger auf den vom Datenpuffer genutzten Speicherbereich und in der Variablen BufLength wird die Zahl der Bytes im Puffer geliefert. Der Parameter User wird hier nicht genutzt; mit ihm kann bei Bedarf eine vom Anwender gewünschte Information transportiert werden; User wird in der Funktion BASS_RecordStart gesetzt. Um auf die einzelnen Samples gezielt und komfortabel zugreifen zu können, haben wir eine eigene Speicherstruktur im Typ TStereoBuffer definiert, einem Array [0..44100] vom Typ TStereoSample. Mit der Prozedur CopyMemory wird der Inhalt des übergebenen Buffers in diesen LocalBuffer kopiert, so dass wir auf die Samples in Form einer zweidimensionalen Feldstruktur zugreifen können.

Die Callback-Routine wird ereignisgesteuert immer dann aufgerufen, wenn ein gefüllter Buffer zur Verfügung steht. Die Default-Länge des Recording-Buffers liegt bei 100 ms entsprechend einer Länge von $100 \text{ ms} * 44100 \text{ Samples/s} = 4410 \text{ Samples} * 2 \text{ Kanäle} * 2 \text{ Bytes} = 17640 \text{ Bytes}$. Allerdings bleibt die tatsächliche Länge des Buffers nicht völlig konstant, weil Windows die Callback-Routine über die Zeit steuert und abhängig von der momentanen Auslastung des Rechners Ereignisse nicht punktgenau eintreffen. Darauf muss man sich bei der Auswertung der Daten einstellen. Der LocalBuffer hat eine Länge von $44100 * 4 = 176400 \text{ Bytes}$; gültig sind davon aber natürlich nur die tatsächlich belegten BufLength Bytes. Die Periodendauer des Datenabrufs wird in der Prozedur RecordButtonClick auf 200 ms eingestellt; das entspricht einer BufLength von rund 35280 Bytes.

Callback-Funktionen müssen einen booleschen Ergebnistyp haben. Über diesen Rückgabewert wird gesteuert, ob die Aufnahme fortgesetzt (true) oder abgebrochen (false) werden soll. Die Aufnahme wird fortgesetzt, solange die Callback-Routine das Ergebnis true zurückgibt:

```
Result := true;  
  
end; // function RecordingCallback
```

Innerhalb unserer Callback-Ereignisroutine müssen wir die eigentliche Auswertung der übergebenen Samplewerte vorsehen. Dafür gibt es verschiedene Möglichkeiten. Z.B. kann das aufgenommene Signal in eine Wave-Datei geschrieben und gespeichert werden. Wie das geht, wird im Beispiel RECORDTEST vorgemacht, das dem BASS-Downloadpaket beiliegt.

Die BASS.DLL stellt eine Reihe von Routinen zur Verfügung, mit denen Parameter des Signals ausgewertet werden können. Besonders einfach ist es, den Level der Samples mit der Funktion BASS_ChannelGetLevel festzustellen, die den Spitzenwert (Peak-Level) des gerade aufgezeichneten Signals liefert. Der Funktion muss als einziger Parameter ein Handle für den ausgewählten Recording-Kanal übergeben werden. Im 16 Bit PCM-Code kann der positive oder negative Maximalwert von $2^{15} = 32768$ erreicht werden. Der Pegel des linken Stereokanals wird im LoWord des Rückgabewerts transportiert, der Pegel des rechten Kanals im HiWord.

```
var  
    Level : Integer;  
    LeftLevel : Word;  
    RightLevel : Word;  
  
begin  
    Level := BASS_ChannelGetLevel(RecChannel);  
    LeftLevel := LoWord(Level);
```

```

    RightLevel := HiWord(Level);
    Form1.ProgressBar1.Position := Trunc(100*RightLevel/cMaxAudio);
end;

```

Im Beispielprogramm wird ein ProgressBar als simple Aussteuerungsanzeige zweckentfremdet. Dazu wird der Peak-Level auf den Maximalwert (32768) normiert und in Prozent umgerechnet.

Da wir auf den Samplepuffer zugreifen können, sind aber auch eigene Auswertungen möglich – hier beginnt die eigentliche Arbeit bei der Entwicklung eigener Messprogramme. Ein simples Beispiel soll zeigen, mit dem wir den RMS-Pegel des aufgenommenen Signals aus dem LocalBuffer ermitteln, soll zeigen, was gemeint ist:

```

var
    LocalBuffer: TStereoBuffer;
    Root, Mean, Square: extended;
    RMSLevel: integer;
    i: cardinal;

begin
    Square := 0;
    for i := 0 to Pred(BufLength div 4) do begin
        Square := Square + LocalBuffer[i,Le] * LocalBuffer[i,Le];
    end;
    Mean := Square/(BufLength div 4);
    Root := Sqrt(Mean);
    RMSLevel := Round(100*Root/cMaxAudio);
    Form1.ProgressBar1.Position := RMSLevel;
end;

```

Die Routine bildet – hier für den linken Aufnahmekanal - die Quadrate der einzelnen Samples, was zu einer Gleichrichtung des Signals führt, da Quadrate immer positiv sind. Die Quadratwerte (Squares) werden aufsummiert, der Durchschnitt (Mean) gebildet und die Wurzel (Root) gezogen – voila: Root-Mean-Square. Nach dem gleichen Zugriffsschema können jetzt natürlich auch Frequenz-Auswertungen, Fourier- und Korrelations-Analysen – um nur Beispiele zu nennen – ausgeführt werden.

[Top](#)

6. Pushing oder Pulling?

Die Nutzung einer Callback-Routine stellt ein Push-Verfahren dar, bei dem wir eine regelmäßige automatisierte Zustellung von Daten organisieren. Das Betriebssystem liefert fortlaufend Puffer voller Samples innerhalb einer vorbestimmten Zeitspanne. Wir müssen diese Daten nur zur rechten Zeit abnehmen und ihre Speicherung oder Weiterverarbeitung organisieren. Die BASS.DLL bietet allerdings auch noch eine zweite Möglichkeit, bei der in einem Pull-Verfahren die Daten gezielt abgerufen werden. Ein solcher Abruf könnte zum Beispiel von einer Timer-Komponente oder auch von einem Button gesteuert werden. Dazu dient die Funktion BASS_ChannelGetData, die entweder einen vordefinierten Puffer mit Samples gefüllt zurück gibt oder aber eine FFT-Auswertung des aufgezeichneten Signals.

[Top](#)

7. Aufnahme starten und stoppen

Der Aufnahmeprozess wird mit der Routine `BASS_RecordStart` eingeleitet. Diese Funktion erwartet insgesamt fünf Parameter: die gewünschte `SampleRate` (z.B. 44100 für CD-Qualität), Anzahl der Kanäle (z.B. 2 für Stereo), Speicheradresse der Callback-Funktion und eine Userkennung. Die Adresse der Callback-Funktion wird mit dem `@`-Operator ermittelt. Außerdem wird der Funktion ein Flag übergeben, mit dem eine Reihe von Optionen gesteuert werden kann. Für unseren Zweck ist interessant, dass der `HiWord`-Anteil dieses `DWord`-Flags zur Einstellung der Lauflänge des Recording-Buffers (in Millisekunden) verwendet werden kann. Die Flag-Variable ist vom `Double-Word`-Typ (`DWord`) und daher 4 Bytes lang. Die gewünschte Periodendauer für die Abtastung wird mit der Funktion `DWord := MakeLong (LoWord,HiWord)` in die oberen zwei Bytes des Flags geschrieben. Dabei beträgt die minimale Zeit zwischen zwei Funktionsaufrufen 10 ms, die maximale Periode ist 500 ms. Wenn die angegebene Periodendauer außerhalb dieses Bereichs liegt, wird sie automatisch gekappt. Der Defaultwert beträgt 100ms.

```
procedure TForm1.RecordButtonClick(Sender: TObject);  
  var  
    Flag: DWord;  
  begin  
    //          ---LoWord---   ---HiWord---  
    Flag := MakeLong(c16BitAudio,cRecordingTime);  
  
    RecChannel := BASS_RecordStart (cSampleRate,  
                                   cNumChannels,  
                                   Flag,  
                                   @RecordingCallback,  
                                   cDefaultUser);  
  
  end; // procedure RecordButtonClick
```

Die Funktion `BASS_RecordStart` gibt im Erfolgsfall einen Handle für die gestartete Aufnahme zurück, der für weitere Zugriffe auf dieses Prozess in der globalen Variablen `RecChannel` gespeichert wird. Dieser Handle wird auch benötigt, um den Aufnahmeprozess zu stoppen. Dazu dient die Funktion `BASS_ChannelStop`, die als einzigen Parameter einen Handle für die zu stoppende Aufnahme erwartet. Im Erfolgsfall gibt die Funktion den booleschen Wert `true` zurück, im Fehlerfall `false`; der Rückgabewert wird hier nicht ausgewertet.

```
procedure TForm1.StopButtonClick(Sender: TObject);  
  begin  
    BASS_ChannelStop(RecChannel);  
  end; // procedure StopButtonClick
```

[Top](#)

8. Zum Schluss: Ein einfacher StatusBar

```
procedure TForm1.UpdateStatusInfo;  
begin  
  with InputChannelsComboBox do begin  
    StatusBar1.SimpleText := 'Aufnahme-Kanal: ' +  
                               Items[ItemIndex];  
    StatusBar1.Enabled := true;  
  end; // with  
end; // procedure UpdateStatusInfo
```

Das Beispielprogramm soll zeigen, wie die Aufnahme und Auswertung von Audiosignalen über die Soundkarte mit Hilfe der BASS.DLL im Prinzip funktioniert. Deshalb wurde hier auf Bedienungskomfort und auch auf die Auswertung möglicher Fehlerwerte – teils ziemlich großzügig – verzichtet. Das Beispielprogramm läuft allerdings auch so recht stabil. Einzige Reminiszenz an den Komfort ist eine kleine Statuszeile, die angibt, welche Signalquelle gerade ausgewertet wird.



[Download dieses Textes als PDF-Datei \(Druckfassung\)](#)



[Download des Delphi-Quelltextes für das Beispielprogramm](#)

[Top](#)

(c) Michael Gaedtke, Im Püllenkamp 2, D-41462 Neuss, Germany