WaveTacer-Programmierung

OXYGENIC

COLLABORATORS			
	TITLE : WaveTacer-Programmierung		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	OXYGENIC	August 27, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

WaveTacer-Programmierung

1.1 Inhaltsverzeichnis

```
                    W a v e T r a c e r   D S  - Programmierung & Formate
          ---------------------------------------------------

Inhaltsverzeichnis:
```

1.2 einleitung

```
  Einleitung
  ----------
```

Der WaveTracer ist ein Soundtool, das Aufgrund seines echten modularen Aufbaus sehr flexibel ist und sich für unkompliziert zu realisierende Erweiterungen hervorragend eignet.
  "Echter modularer Aufbau" bedeutet, das die verschiedenen wählbaren Module wirklich als kleine, externe Programme vorliegen und nicht etwa alle im eigentlichen Hauptprogramm integriert sind, welches dann natürlich sehr groß wäre (die Betonung liegt auf "SEHR", da dann mit einer Größenordnung im MByte-Bereich zu rechnen wäre; diese würden dann aber hauptsächlich nur aus Programmcode bestehen, der nur wenig oder nur kurzzeitig benutzt würde, sonst aber nur wertvollen Speicher verschwendet).
  Weiterhin wird hier der Aufbau der vom WaveTracer DS unterstützten Formate beschrieben, um eine Verwendung in anderen Programmen zu ermöglichen.

Sollte jemand Module für den WaveTracer DS entwickelt haben, besteht die Möglichkeit, diese zusammen mit dem WaveTracer DS Komplettpaket vertreiben zu lassen. Voraussetzung hierfür ist die Einhaltung aller Copyright-Bedingungen und eine korrekte Programmierung. Genaue Informationen zu Copyright und Programmierung finden sich weiter unten.
  Wer also seine Module für das Komplettpaket zur Verfügung stellen will, sendet diese einfach an den Autor. Die Kontaktadresse befindet sich in der Hauptdokumentation "WaveTracer.Guide".

Über eine Umsetzung der mitgelieferten Beispiele und Include-Dateien in andere Programmiersprachen wäre ich auch sehr erfreut. Ich würde mich dafür natürlich erkenntlich zeigen.

1.3 copyright

```
  Copyright
  ---------
```

Der WaveTracer DS ist © by VIRTUAL WORLDS PRODUCTIONS & OXYGENIC. Alle Rechte am WaveTracer DS, den WaveTracer-spezifischen Soundformaten und den Formaten der Modulschnittstellen verbleiben beim Autor.

Die Benutzung der Soundformate und der Modulschnittstellen für die Programmierung von Lade-/ Speicher-/ Soft-/ Effektmodulen ist erlaubt. Hierfür werden keine Lizenzgebühren erhoben oder sonstige Einschränkungen gemacht.

Die Benutzung der Modulschnittstellen und der Module von anderen Programmen als von Modulen aus, speziell von einem anderen Hauptprogramm als dem WaveTracer, bedarf einer schriftlichen Einverständniserklärung des Autors.

Die Benutzung der WaveTracer-spezifischen Soundformate ist erlaubt. Es werden sowohl für die reine Benutzung der Formate als auch für Programme/ Programmteile, die diese Formate laden oder speichern können, keine Lizenzgebühren erhoben. Ausnahme ist das CineData-Format, für das eine andere Regelung gilt. Diese ist in der Dokumentation "CineTracer.Guide" ausführlich dargelegt.

Diese Dokumentation stellt einen Teil der Gesamtdokumentation dar, so das ebenfalls alle anderweitig aufgeführten Erklärungen und Bedingungen für diesen Teil der Dokumentation gelten, sofern sie die hier getroffenen Regelungen nicht aufheben oder ihnen widersprechen.

Für jegliche Benutzung irgendeines Teils dieses Softwarepaketes oder der hier beschriebenen Formate auf einem anderen Computersystem als dem Amiga® oder einem Amiga®-kompatiblen System oder einem Emulator gilt, das dafür die schriftliche Einverständniserklärung des Autors vorliegem muß!

1.4 iff

```
                   Das Prinzip der IFF-Formate
  ---------------------------
```

Das IFF-Format (Interchanging FileFormat) ist trotz seines, für das Computerzeitalter biblischen Alters, nach wie vor das flexibelste. Das liegt sicher daran, das hier, ohne groß rumzuzaubern oder sich mit antiken Problemen beim Datenformat rumzuärgern, ganz einfach Erweiterungen vorgenommen werden können. Dies ist in der Struktur dieses Formates direkt vorgesehen, wie im folgenden gezeigt werden soll. Die Beschreibung der Formate wird durch die Angabe von Records, wie sie in
Pascal
üblich sind, unterstützt.

Die Werte für diverse Flags existieren als Konstante (Const) und befinden sich in der mitgelieferten und weiter unten detailliert beschriebenen WaveTracer-Include-Datei.

Der Grundaufbau des IFF ist eigentlich immer gleich. Als erstes steht in der Datei immer ein Longword (bzw. ein 4-stelliger String) mit dem Inhalt

'FORM'. Das ist die Kennzeichnung für das IFF-Format. Das darauffolgende Longword gibt die Länge des Files ab dieser Stelle an (oder anders formuliert: die Gesamtlänge des Files minus 8).
  Danach folgt die Kennung für die Art des Files. Das ist wiederum ein Longword (z.B. 'ILBM' für Bilder, 'FTXT' für Texte, '8SVX' für Sounds,...).

```
type IFF_Header=record
   IFF_ID          :string[4];      IFF-Kennung 'FORM'
   Filelength      :long;           Dateigesamtlänge minus 8
   IFF_Type        :string[4];      Art des IFF
end;
```

Darauf folgen mehrere (verschiedene) Chunks, die alle denselben Grundaufbau besitzen und die eigentlichen Informationen beinhalten.
Man ist ohne weiteres in der Lage, neue, eigene Chunks zu kreieren und in die Datei einzufügen, ohne das irgendwelche Probleme mit anderer Software auftreten. Diesen Chunks ist nur ein Name zu geben, der noch nicht als Chunk-Bezeichnung existiert. Es sollte hier aber beachtet werden, das Software, die den neuen Chunk nicht kennt, auch die darin enthaltenen Daten nicht verarbeiten kann.
  Am Anfang eines Chunks steht wieder eine Kennung (z.B. 'VHDR' für Voice-Header, 'BMAP' für BitMap, 'CRNG' für ColorRange,...). Darauf folgt wieder ein Longword mit der Länge des Chunks ab diese Stelle (oder: Chunk-Gesamtlänge minus 8). Nun folgen die eigentlichen Daten des Chunks. Diese sind genau so groß, wie in der Chunklänge angegeben.

```
type IFF_Chunk=record
   Chunk_ID        :string[4];      Chunk-Kennung
   Chunk_Length    :long;           Gesamtlänge des Chunks minus 8
   Chunk_Data      :array [1..Chunk_Length] of byte;
                                    Hier folgen die eigentlichen Daten
                                    des Chunks, der so lang ist, wie
                                    unter Chunk_Length angegeben
end;
```

Es ist ganz klar zu erkennen, warum dieses Dateiformat kompatibel sein muß: Findet ein Programm einen unbekannten Chunk, stellt es fest, wie lang er ist, springt in der Datei entsprechend weit vorwärts und findet an dieser Stelle den nächsten Chunk. Es ist dabei nur zu beachten, das Programme Informationen in unbekannten Chunks auch nicht nutzen können. Auch gibt es Probleme, wenn zwei Programme unabhängig voneinander einen Chunk mit dem selben Namen aber anderem Inhalt erzeugen. Es ist also darauf zu achten, das keine Chunks, die hier beschrieben werden, unter dem selbem Namen neu kreiert werden. Diese Auflistung der existierenden Soundformate-Chunks erhebt jedoch keinen Anspruch auf Vollständigkeit. Sie soll vielmehr einen Überblick über die wichtigsten vorkommenden Chunks geben.

Der genaue Aufbau des WaveTracer-Sound-IFF wird im folgenden beschrieben, ohne die hier genannten Fakten zu wiederholen, da das IFF-Prinzip überall gleich ist.

1.5 8svx

Das IFF-8SVX-Format

Dieses Format ist kein WaveTracer-spezifisches Soundformat. Vielmehr ist es fast genauso alt, wie der Amiga® selbst. Es enthält die Daten in 8 Bit Auflösung. Ursprünglich konnte hiermit immer nur ein Kanal abgespeichert werden. Der AudioMaster® schließlich führte einen neuen Chunk ein, der eine Kennung enthielt, welche Kanäle von maximal zwei Möglichkeiten im File vorhanden sind. Der WaveTracer DS unterstützt diesen Chunk ('CHAN') mit einer entsprechenden Erweiterung auf 6 Kanäle
Der Header:

```
type IFF_Header=record
   IFF_ID          :string[4];     ='FORM' IFF-Kennung
   Filelength      :long;           Dateilänge minus 8
   IFF_Type        :string[4];     ='8SVX' IFF-Typ
end;
```

Danach folgt üblicherweise der VHDR-Chunk:

```
type IFF_Chunk=record
   Chunk_ID          :string[4];  ='VHDR' - VoiceHeaDeR
   Chunk_Length      :long;         Länge (20 Bytes)
   OneShotHiSamples  :long;         Anzahl der Bytes, die einmal abgespielt
                                    werden
   RepeatHiSamples   :long;         Anzahl der Bytes, die mehrfach abge-
                                    spielt (geloopt) werden (normaler Loop)
   SamplesPerHiCycle :long;
   SamplesPerSec     :word;         Samplefrequenz, die Umrechnung in die
                                    Amiga-Playrate geschieht wie folgt:
                         Playrate:=round(10000000/(SamplesPerSec*2.79365))
   CtOctave1         :byte;
   SCompression1     :byte;         Kompressionsart:
                                       0 - keine Kompression
                                       1 - COMPRESSION_FIBONACCI_DELTA,
   FixedVolume1      :byte;         Lautstärke im Bereich von 0 bis 64;
   CtOctave2         :byte;
   SCompression2     :byte;         Kompressionsart, siehe oben
   FixedVolume2      :byte;
end;
```

Weitere, vom WaveTracer DS unterstützte Chunks:

```
type IFF_Chunk=record
   Chunk_ID          :string[4];    ='CHAN' - CHANnels
   Chunk_Length      :long;          Länge: 4 Bytes
   UsedChannels      :long;          Gibt die vorhandenen Kanäle an:
                                        1: SL -  CH_SLEFT,
                                                 Surround/Surround Links
                                        2: L -   CH_LEFT, Links
                                        4: R -   CH_RIGHT, Rechts
                                        8: C -   CH_CENTER, Center
                                       16: SR -  CH_SRIGHT,
                                                 Surround Rechts
                                       32: Sub - CH_SUB,
                                                 Subwoofer-Kanal
```

```
   end;
   Die Flags der UsedChannels-Variable sind ziemlich ungeordnet. Das
   Problem besteht darin, das sie mehrfach angepaßt werden mußten: beim
   AudioMaster® auf zwei Kanäle, beim WaveTracer 1.x auf 4 Kanäle und
   schließlich beim WaveTracer DS durch die Einführung des DTS® /
   Dolby-AC-3® Formats auf 6 Kanäle. Daher sollten beim WaveTracer
   zweckmäßigerweis die HISX-Soundformate verwendet werden, da diese
   besser für viele Kanäle geeignet sind und zusätzlich WaveTracer-
   spezifische Informationen wie z.B. über den SoundMode enthalten.

   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='ADSR' - Attack, Decay, Sustain,
                                                 Release
      Chunk_Length      :long;          Länge: 16 Bytes
      Attack            :long;          Position in Samples
      Decay             :long;              "     "    "
      Sustain           :long;              "     "    "
      Release           :long;              "     "    "
   end;
   Ist ein Wert für Sustain angegeben, so ist das Sample ab dieser Stelle
   zu loopen. Sind Sustain und Release angegeben, so sind die Daten in
   diesem Bereich zu loopen (Sustain-Release-Loop, wird z.B. mit dem
   Funktionsmodul "SetLoop" eingestellt).

   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='ACHN' - Alpha-Channel
      Chunk_Length      :long;           Länge: Samples div 20 * Anzahl
                                                vorhandener Kanäle
      Daten             :array [1..Chunk_Length] of byte
   end;
   Dieser Chunk enthält die Daten für den Alphakanal in Form von 8Bit-
   Daten. Diese sind in der gleichen Reihenfolge wie die Daten im BODY-
   Chunk abgespeichert. Die Besonderheit beim Alpha-Kanal ist, das
   dieser nur 1/20 so lang ist wie die eigentlichen Sampledaten.

Zum Schluß folgen dann die eigentlichen Sounddaten:

   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='BODY'
      Chunk_Length      :long;          Länge: Samplelänge mal Anzahl Kanäle
      ... Danach folgen die Daten pro Kanal, z.B.:
                  1. Daten Kanal L
                  2. Daten Kanal R
      Die Reihenfolge der Kanäle, in der hier abgelegt wird, ist L, R, C,
      SL, SR und zum Schluß Sub.
   end;
```

1.6 16sv

```
  Das IFF-16SV-Format
  -------------------

Dieses Format ist ebenfalls kein WaveTracer-spezifisches Soundformat. Es
ist aus dem 8SVX-Formates hervorgegangen und wurde meines Wissens nach von
```

Richard Körber für das Programm SoundBox® entwickelt. Bis auf die Daten im BODY-Chunk sind das 8SVX- und das 16SV-Format gleich. Bei Länge und Loop-Offset ist zu beachten, das hier in Samples und nicht in Bytes gerechnet wird. Um also die Länge in Bytes zu erhalten, ist einfach der Sample-Wert mit zwei zu multiplizieren.
Die Sampledaten sind in normalen Words abgespeichert.

Da zum 8SVX-Format keine großen Unterschiede bestehen, wird das 16SV-Format hier nur noch kurz beschrieben

```
type IFF_Header=record
   IFF_ID          :string[4];     ='FORM'
   Filelength      :long;           Dateilänge minus 8
   IFF_Type        :string[4];     ='16SV'
end;

type IFF_Chunk=record
   Chunk_ID          :string[4];  ='VHDR'
   Chunk_Length      :long;         Länge
   OneShotHiSamples  :long;         Anzahl der Bytes, die einmal abgespielt
                                    werden
   RepeatHiSamples   :long;         Anzahl der Bytes, die mehrfach abge-
                                    spielt (geloopt) werden (normaler Loop)
   SamplesPerHiCycle :long;
   SamplesPerSec     :word;         Samplefrequenz, die Umrechnung in die
                                    Amiga-Playrate geschieht wie folgt:
                        Playrate:=round(10000000/(SamplesPerSec*2.79365))
   CtOctave1         :byte;
   SCompression1     :byte;         Kompressionsart:
                                       0 - keine Kompression
                                       4 - COMPRESSION_DELTA,
                                           Delta-1-Kompression
                                       8 - COMPRESSION_DELTA_2,
                                           Delta-2-Kompression
   FixedVolume1      :byte;         Lautstärke im Bereich von 0 bis 64;
   CtOctave2         :byte;
   SCompression2     :byte;         ungenutzt, da die Delta-Kompres-
                                    sionen immer auf alle Kanäle
                                    angewandt werden
   FixedVolume2      :byte;
end;

type IFF_Chunk=record
   Chunk_ID          :string[4];    ='CHAN' - CHANnels
   Chunk_Length      :long;          Länge: 4 Bytes
   UsedChannels      :long;          Gibt die vorhandenen Kanäle an
end;

type IFF_Chunk=record
   Chunk_ID          :string[4];    ='ADSR' - Attack, Decay, Sustain,
                                               Release
   Chunk_Length      :long;          Länge: 16 Bytes
   Attack            :long;          Position in Samples
   Decay             :long;              "     "    "
   Sustain           :long;              "     "    "
   Release           :long;              "     "    "
end;
```

```
type IFF_Chunk=record
   Chunk_ID          :string[4];    ='ACHN' - Alpha-Channel
   Chunk_Length      :long;           Länge: Samples div 20 * Anzahl
                                             vorhandener Kanäle
   Daten             :array [1..Chunk_Length] of byte
end;

type IFF_Chunk=record
   Chunk_ID          :string[4];    ='BODY'
   Chunk_Length      :long;          Länge: Samplelänge mal 2
                                            mal Anzahl Kanäle
   Danach folgen die Daten pro Kanal in Words
end;
```

1.7 xxsx

```
                Das IFF-16SX- und 24SX-Format
  ----------------------------
```

Das 16SX und das 24SX-Format sind WaveTracer-spezifische Formate und © by VIRTUAL WORLDS PRODUCTION & OXYGENIC.

Hier noch ein paar Worte zu dem (eigentlich bisher nur bei Microsoft® so üblichen) Chaos mit den ADSR-Chunks, bei denen die Offsets einmal in Samples angegeben werden (beim 16SV-Format) und einmal in Bytes (bei 16SX und 24SX). Diese Werte sollten ursprünglich alle die Einheit Samples haben, so das sie mit einem Faktor (2 oder 4) multipliziert werden müßten um die Offsets in Bytes zu erhalten. Leider hat sich in die Speicherroutine des WaveTracer DS ein Bug eingeschlichen, der das ganze dann so verkorkst hat. Und leider habe ich den erst so spät bemerkt, das eine Änderung nicht mehr möglich war, da das Chaos sonst nur noch größer geworden wäre.
Somit verdanken wir also den allseits bekannten murphy´schen Gesetzen wieder eine neue Komplikation. Jetzt bin aber erstmal wieder ich am Zug und ich werde eine kompatible Möglichkeit suchen, die da wieder Ordnung reinbringt.

Da der Amiga standardmäßig nur eine 4-kanalige 8 Bit-Soundausgabe (bzw. mit ein paar Tricks eine 2-kanalige 16 Bit-Soundausgabe) besitzt, existieren keine offiziellen Commodore®-Soundformate für mehr als 8 Bit.
Das 16SX-Soundformat ist für eine Sampleauflösung im Bereich von 9 bis 16 Bit vorgesehen und auf die Anforderungen der WaveTracer-Soundmodi zugeschnitten.

Mit dem 24SX-Fomat ist man theoretisch in der Lage, 32-Bit Samples abzuspeichern. Da das menschliche Ohr jedoch nicht in der Lage ist, einen Unterschied zwischen 24 und 32 Bit wahrzunehmen, ist das unnötig.
Selbst die Soundverarbeitung in 24 Bit erfolgt nur in Studios, der eigentliche Tonträger (z.B. Compact Disk) hat dann nur noch eine Auflösung von 16 Bit (HighEnd-Freaks behaupten zwar, das sie einen qualitativen Unterschied zwischen 16 und 20 Bit hören, aber ich vermute, das das auch nur der Fall ist, wenn man ihnen vorher sagt, welche Auflösung sie gerade hören).
Im 24SX-Format werden also Samples von 17 bis 24 Bit abgespeichert.

```
Die ungenutzten, restlichen 8 Bit werden bei der immer benutzten 3Byte-
Kompression nicht mit abgespeichert (Achtung: Es könnte noch Samples geben,
deren 24Bit-Daten mit vollen 32 Bit - also unkomprimiert - abgespeichert sind!).

Der Aufbau der beiden Sampleformate sieht wie folgt aus:

Der Header:

   type IFF_Header=record
      IFF_ID          :string[4];     ='FORM' IFF-Kennung
      Filelength      :long;           Dateilänge minus 8
      IFF_Type        :string[4];     ='16SX' oder '24SX'
   end;

Danach folgt der SXHD-Chunk:

   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='SXHD' - SXHeaDer
      Chunk_Length      :long;          Länge (22 Bytes)
      SampleDepth       :byte;          Auflösung des Samples
                                        bei 16SX: 9 bis 16 Bit sind möglich,
                                           16 aber typisch
                                        bei 24SX: 17 bis 32 Bit möglich,
                                           24 typisch
      FixedVolume       :byte;          gemeinsame Lautstärke aller Kanäle
                                        von 0 bis 64
      Length            :long;          Länge des Sounds in der Einheit
                                        Samples
      PlayRate          :long;          Amiga-Playrate
      CompressionMethod :long;          Kompression, wirkt immer auf alle
                                        vorhandenen Kanäle:
                                          0 - keine
                                          2 - COMPRESSION_3BYTE,
                                               3Byte-Runtime, sollte bei 24SX
                                               IMMER angewendet werden;
                                               der Sound wird in reinen 24
                                               Bit ohne die ungenutzten
                                               8 Bit pro Sample abgespeichert
                                          4 - COMPRESSION_DELTA,
                                               Delta-1-Kompression
                                          8 - COMPRESSION_DELTA_2,
                                               Delta-2-Kompression

      UsedChannels      :byte;          spezifiziert die im File vorhan-
                                        denen Kanäle:
                                           1: L -   CH_LEFT, Links
                                           2: R -   CH_RIGHT, Rechts
                                           4: C -   CH_CENTER, Center
                                           8: SL -  CH_SLEFT,
                                                    Surround/Surround Links
                                          16: SR -  CH_SRIGHT,
                                                    Surround Rechts
                                          32: Sub - CH_SUB, Subwoofer-Kanal
      UsedMode          :byte;          spezifiziert den Soundmode, Kombi-
                                        nationen mehrerer Soundmodi sind
                                        nicht möglich:
                                           1: MD_MONO, Mono
```

```
                                          2: MD_STEREO, Stereo
                                          3: MD_3CH, 3Channels
                                          4: MD_QUADRO, Quadro
                                          5: MD_SURROUND, Dolby Surround®
                                          6: MD_AC3_Sub,
                                             Dolby AC-3® ohne Subwoofer
                                          7: MD_AC3,
                                             Dolby AC-3® mit Subwoofer
      PlayFreq          :long;          Play- bzw. Samplefrequenz, dieser
                                        Wert sollte dem "PlayRate"-Wert im
                                        allgemeinen entsprechen
      Loop              :word;          Anzahl der Wiederholungen, wenn
                                        der SXHD-Chunk in einer
               Animation
                                                        enthalten ist
   end;
```

Der folgende Chunk wird vom WaveTracer immer mit angelegt, ist aber nicht zwingend erforderlich:

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='ADSR' - Attack, Decay, Sustain,
                                                 Release
      Chunk_Length      :long;          Länge: 16 Bytes
      Attack            :long;          Position in Bytes
      Decay             :long;              "     "   "
      Sustain           :long;              "     "   "
      Release           :long;              "     "   "
   end;
```

Soll die Position in der Einheit Samples ermittelt werden, müssen die Werte durch 2 (bei 16SX) bzw. 4 (bei 24SX) dividiert werden.

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='ACHN' - Alpha-Channel
      Chunk_Length      :long;           Länge: Samples div 20 * Anzahl
                                                vorhandener Kanäle
      Daten             :array [1..Chunk_Length] of byte
   end;
```

Dieser Chunk enthält die Daten für den Alphakanal in Form von 8Bit-Daten. Diese sind in der gleichen Reihenfolge wie die Daten im BODY-Chunk abgespeichert. Die Besonderheit beim Alpha-Kanal ist, das dieser nur 1/20 so lang wie die eigentlichen Sampledaten ist (oder in absoluter Länge in Bytes ausgedrückt: 1/40).

Als letztes folgt der BODY-Chunk mit den Sounddaten:

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='BODY' - CHANnels
      Chunk_Length      :long;          Länge: Samplelänge mal Anzahl
                                        Kanäle mal 2 (bei 16 Bit!)
                                        bzw. mal 3 (bei 24 Bit!)
      ... Danach folgen die Daten pro Kanal in Words, z.B.:
                  1. Daten Kanal L
                  2. Daten Kanal R
      Die Reihenfolge der Kanäle, in der hier abgelegt wird, ist L, R, C,
      SL, SR und zum Schluß noch Sub.
```

```
   end;
```

```
  Besonderheiten im SXHD-Chunk
  ----------------------------
```

Es werden hier zwei verschiedene Werte für die Abspielgeschwindigkeit angegeben: PlayRate und PlayFreq. PlayRate entspricht der Amiga-Playrate, die von dessen Audiohardware erwartet wird (und zudem noch vom aktuellen Screenmodus beeinflußt wird). Die PlayFreq ist der wesentlich genauere Wert, der die reale Samplefrequenz angibt. Aufgrund von Rundungsfehlern bei der Unrechnung können die Werte leicht differieren. Für exakte Ergebnisse ist also PlayFreq als Grundlage zu verwenden.

PlayRate und PlayFreq lassen sich nach folgender Formel umrechnen:

$$\text{PlayFreq} = \frac{10000000}{\text{PlayRate} * 2.79365}$$

Der Wert für Loop ist in reinen Samplefiles uninteressant. Ist der SXHD-Chunk allerdings Teil einer
Animation
, so gibt dieser Wert an, wie oft diese wiederholt werden soll.

1.8 hisx

```
                  MEHRERE xxSX-Formate - oder EIN "HISX"-Format?
  ----------------------------------------------
```

16SX und 24SX unterscheiden sich eigentlich nur durch die unterschiedliche Bitbreite (die im SXHD-Chunk angegeben wird) und durch ihren Namen. Es wäre also prinzipiell möglich, sie durch ein einziges Format zu ersetzen, das zudem den vollen Bereich der Sampleauflösungen bis hinunter zu 8 Bit abdeckt.

Das das ohne weiteres möglich ist, wird im
ANIM-Soundformat
gezeigt, bei dem ein vollständig kompatibler SXHD-Chunk für praktisch alle Sampleaufösungen eingesetzt wird.

Die einzig logische Antwort auf dieses Formatechaos wäre also ein xxSX-format, das alle Auflösungen beinhaltet: das HISX-Format.

HISX wird in Kürze beim WaveTracer implementiert und wird auf lange Sicht die beiden Formate 16SX und 24SX ablösen.

Es wäre also sinnvoll, Loader oder andere Programme darauf vorzubereiten. Einzig beim Format-Namen ist eine Ergänzung vorzunehmen: zu "16SX" und "24SX" kommt "HISX". Da die möglichen Auflösungen sich sowieso schon in weiten Grenzen bewegen dürfen (9 bis 16 Bit oder 17 bis 32 Bit), ist das bei vernünftig programmierten Laderoutinen die einzige Änderung, ansonsten ist mit HISX genauso zu verfahren, wie mit 16SX und 24SX.

1.9 iff-besonderheiten

```
  Besonderheiten bei Sound-IFF's, die vom WaveTracer gespeichert wurden
  ---------------------------------------------------------------------
```

IFF-Dateien des WaveTracer DS können mitunter Chunks beinhalten, die eigentlich von völlig anderen IFF-Typen stammen. Diese werden vom WaveTracer mit abgespeichert und entsprechend auch wieder geladen, wenn sie Daten enthalten, die für bestimmte Operationen oder Programmteile nötig sind. Der WaveTracer DS selbst speichert solche Informationen in einer etwas anderen Strukturierung mittels der SpecialData-Struktur bzw. -Liste im RAM.

Momentan sind folgende spezielle Chunks in den IFF-Formaten 8SVX, 16SV, 16SX und 24SX gebräuchlich:

```
  Der DPAN-Chunk
  --------------
```

Dieser Chunk stammt von DeluxePaint® und wird dort bei Animationen verwendet um bestimmte Werte, die für den Anim-Player wichtig sind, zu speichern.
  Beim WaveTracer ist dieser Chunk bzw. die SpecialData-Struktur des Typs SD_ANIMINFO wichtig, wenn ein Sound synchron zu einer Animation abgespielt werden soll, bzw. wenn ein Sound mit Hilfe des AnimInjector-Speichermoduls in ein Animationsfile integriert werden soll.

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='DPAN' - DeluxePaint® ANimation
      Chunk_Length      :long;         = Länge: 8 Bytes
      Version           :word;         = Versionsnummer des Chunks, kann
                                         ignoriert werden
      Frames            :word;         = Anzahl der in der Animation enthal-
                                         tenen Frames (= Einzelbilder)
      FPS               :byte;         = Playrate in der Einheit Frames pro
                                         Sekunde
      pad1,pad2,pad3    :byte;         = Füllbytes;
   end;
```

```
  Der ANIM-Chunk
  --------------
```

Dieser Chunk ist ein WaveTracer-spezifischer Chunk, der den Pfad zu einer IFF-ANIM-Datei beinhaltet. ANIM-Chunks kommen in Dateien vor, die im Zusammenhang mit einer Animation erzeugt wurden. Der WaveTracer nutzt den Pfad der Animation, um deren Frames im "Anim-Frames"-Fenster darzustellen.

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='ANIM' - ANIMationsdatei
      Chunk_Length      :long;         = Länge: 200 Bytes;
      AnimPath          :string[200];  = Pfad zur IFF-ANIM-Datei
   end;
```

```
  Der WTSZ-Chunk
  --------------
```

```
Dieser Chunk wird abgespeichert, wenn eine szenenweise Soundbearbeitung
stattfindet (z.B. bei sehr großen Animtionen). Die hier enthaltenen Daten
werden vom WaveTracer DS in einer SDBodySzene-Struktur gespeichert.

   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='WTSZ' - ANIMationsdatei
      Chunk_Length      :long;         = Länge: 208 Bytes;
      Name              :string[200];  = der Name der Szene (momentan
                                         unbenutzt)
      StartFrame        :long;         = erstes Frame der Szene
      EndFrame          :long;         = letztes Frame der Szene
   end;
```

1.10 anim

```
                  Das ANIM-Soundformat
  --------------------

Mit der Einführung des CineTracers® und des CineData-Lademoduls, die eine
Vertonung von Animationen ermöglichen, war es notwendig geworden, das nor-
male ANIM-Format so zu erweitern, das Sampledaten mit abgespeichert werden
können.
  Als Header für alle wichtigen Informationen wird hierbei wieder der
bereits bekannte
                SXHD-Chunk
                 verwendet. Die eigentlichen Sample-
informationen sind dann in der gleichen Form, wie weiter
                oben
                 schon
beschrieben, in den SBDY-Chunks ("SoundBoDY") gespeichert. Diese mußten so
benannt werden, da der Chunkname "BODY" in Animationen bereits vergeben ist
und die Imagedaten enthält.
  Eine vom WaveTracer vertonte Animation hat dann folgendes Format:

FORM ANIM
.  FORM ILBM
.  .  BMHD
.  .  DPAN - der
                DeluxePaintANim-Chunk
                 wird vom WaveTracer eingefügt,
             sofern er nicht schon vorhanden ist
.  .  SXHD - der
                SXHD-Chunk
                 mit allen wichtigen Definitionen
.  .  SBDY - die Sampledaten für das erste Bild
.  .  ANHD
.  .  CMAP
.  .  CAMG
.  .  BODY
.  FORM ILBM
.  .  ANHD
.  .  SBDY - die Sampledaten für das zweite Bild
.  .  DLTA
```

```
  * * *

.  FORM ILBM
.  .  ANHD
.  .  SBDY - die Sampledaten für das letzte Bild
.  .  SBDY - "überschüssige" Sampledaten, die während der Aus- oder
             überblendung abgespielt werden
.  .  DLTA


Der SXHD-Chunk weist hier folgende Besonderheiten auf:
   - die Bitbreite ist - entsprechend der
                HISX-Spezifikation
                 - nur aus der
     Variablen "SampleDepth" zu erkennen
   - der Wert "Length" enthält die Länge der Sampledaten für ein einzelnes
     Frame und nicht wie sonst für das komplette Sample. Diese Länge
     entspricht aber nur der bei einer festen Playrate zu erwartenden
     Länge. Differiert die Anzeigedauer eines Frames innerhalb einer
     Animation, so differiert natürlich auch die Länge des SBDY-Chunks!
     In diesem Fall ist der Wert von "Length" natürlich nutzlos und kann
     =0 sein.
   - es darf hier KEINE der vorgesehenene Kompressionsmethoden verwendet
     werden, da diese 1. die Playroutine enorm bremsen würden und 2. bei
     Dolby-Surround®-codierten Samples die Hintergrundinformationen zer-
     stören!!!
   - als "UsedMode" ist momentan nur MD_MONO oder MD_STEREO möglich
   - der Wert für "Loop" hat hier - im Gegensatz zu den reinen
                Samplefiles
                     - eine Bedeutung. Ist Loop=0 so liegt eine normale Animation  ↩
                        vor. Ist
     der Wert >0 so gibt er an, wie oft die Animation nach dem ersten mal
     erneut abgespielt werden soll. Weiterhin ist dann darin die
     Information enthalten, das die Grafikdaten der Animation ebenfalls
     einen Loop enthalten, d.h. das die ersten beiden Frames als zusätz-
     liche Framedaten am Ende der Animation wiederholt werden

Der SBDY-Chunk enthält die Sampledaten für 1 oder 2 Kanäle für die spezi-
fizierte Anzeigedauer eines Bildes. Um einen gleichmäßigen Animationsablauf
zu erhalten, kann die ANIM-Playrate mittels der Abspieldauer der Sample-
daten für das jeweilige Bild zu synchronisiert werden! Hierbei ist aber zu
beachten, das die Abspielgeschwindigkeit einer Amiga-Playrate bei unter-
schiedlichen Screenmodi differieren kann und eventuell zu modifizieren ist.
  Im letzten Bild einer Animation können zwei SBDY-Chunks vorkommen. Der
erste Chunk ist - wie bekannt - für das Bild vorgesehen. Der zweite Chunk
enthält Sampledaten, die z.B. bei der Berechnung eines Surround-Halls ent-
standen sind. Diese Daten sind am Ende der Animation während der Ausblendung
aus der Animation oder während der Überblendung zur nächsten Animation
abzuspielen.

ACHTUNG! Da es in Zukunft Programme geben kann, die Animationen mit Sound
korrekt verarbeiten, kann es passieren, das eine Animation, die z.B. aus
zwei anderen zusammengesetzt wurde, mitten im File zwei SBDY-Chunks hinter-
einander besitzt. Hier kann bei der Überblendung z.B. folgendermaßen
verfahren werden: Der 2. SBDY-Chunk ist auf den bereits zuvor verwendeten
Kanälen abzuspielen (z.B. A und B). Die Sounddaten des neuen Frames sind
```

sofort abzuspielen (auf den Kanälen D und C). Das ergibt eine akustische Überblendung, wie sie z.B. aus Kinofilmen bekannt sein sollte.
  Wie hier letztendlich verfahren wird ist mir egal, das Ergebnis sollte aber so aussehen, das die beiden sich überlappenden Sounds auch zusammen abgespielt werden!

1.11 wtpl

```
                  Das IFF-WTPL-Format
  --------------------
```

Die WTPL-Dateien enthalten keinerlei Sound- bzw- Sampledaten. Hierbei handelt es sich vielmehr um die Playlisten- und Time-Pattern-Informationen des Playlisten-Editors bzw. des Fensters "Time-Pattern". Die zwei möglichen Chunks müssen weder in der unten beschriebenen Reihenfolge auftreten, noch ist festgelegt, das beide vorhanden sein müssen. Theoretisch ist es sogar möglich, das keiner der Chunks "PATT" oder "ENTR" vorhanden ist, wenn z.B. eine leere Playliste abgespeichert wurde.
  Ergibt sich bei einem der Chunks eine Gesamtlänge, die kein Vielfaches von 4 ist, so werden am Ende entsprechend Füll-Bytes angehängt.

Der Header:

```
   type IFF_Header=record
      IFF_ID          :string[4];     ='FORM' IFF-Kennung
      Filelength      :long;          = Dateilänge minus 8
      IFF_Type        :string[4];     ='WTPL'
   end;
```

Der PATTern-Chunk enthält die Daten der Patternliste

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='PATT' - Patternlisten-Informationen
      Chunk_Length      :long;         = Patterns: Chunk_Length div 38
      Daten             :array [1..Patterns] of PlayListPattern
   end;

   Die eigentlichen Pattern-Informationen sind in folgender Struktur fest-
   gelegt:
   type PlayListPattern=record
      BeginOffset,EndOffset :long;       = Anfangs- und Endoffset des
                                           Patterns vom Anfang des Samples
                                           gerechnet
      Name                  :string[30]; = Name des Patterns
   end;
```

Der ENTRy-Chunk enthält die eigentlichen Playlisten-Daten. Hier ist - im Gegensatz zum PATTern-Chunk die Reihenfolge wichtig!

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='ENTR' - Patternlisten-Informationen
      Chunk_Length      :long;         = Entries: Chunk_Length div 56
```

```
      Daten             :array [1..Entries] of PlayListEntry
   end;

   Die Playlisten-Informationen sind in der PlayListEntry-Struktur
   enthalten:
   type PlayListEntry=record
      BeginOffset,EndOffset :long;       = Anfangs- und Endoffset des
                                           Playlisten-Eintrags vom Anfang
                                           des Samples gerechnet
      Rate                  :long;       = Amiga-Playrate
      Delay,Time            :long;       = derzeit unbenutzt und daher =0
      Loop                  :integer;    = Anzahl Loops
      VolumeL,VolumeR       :integer;    = Lautstärke Links/Rechts
      Name                  :string[30]; = Name des Playlisten-Eintrags

Zusätzlich kann eine WTPL-Datei noch einen
                ANIM-Chunk
                 besitzen, der
den Dateinamen einer IFF-ANIM Datei angibt. Dieser wird benötigt, um
Animationsframes im "Anim-Frames"-Fenster darzustellen.
```

1.12 cinm

```
                  Das IFF-CINM-Format
  -------------------
```

Für das CineData-Format wurden abweichende Copyright- und Lizenzregelungen geschaffen, die in der Dokumentation "CineTracer.Guide" ausführlich dargelegt werden und unbedingt beachtet werden müssen.

Die CINM-Dateien enthalten - ähnlich wie die WTPL-Files - keinerlei Sounddaten. Bei diesen sind die Positionen verschiedener Objekte (z.B. aus Raytracing-Programmen) enthalten. Der WaveTracer ist mittels des CineData-Lademoduls in der Lage, aus diesen Daten einen Sound zu generieren, der perfekt zu einer (mit dem Raytracer erzeugten) Animation paßt.

Die Objekt-Koordinaten, die im FRDT-Chunk gespeichert sind, stehen in Relation zur Position des Hörers, der die Koordinaten 0,0,0 hat. Positionen vor, rechts und über dem Hörer besitzen positive Koordinatenwerte, Positionen links, hinter und unter dem Hörer besitzen dementsprechend negative Werte.

Die Lautsprecher wiederum besitzen Positionen um den Hörer herum. Deren Koordinaten lassen sich aus dem Threshold-Wert des CNHD-Chunks errechnen.

Der Bereich, in dem noch Geräusche hörbar sind, endet in einer räumlichen Entfernung vom jeweiligen Lautsprecher, die durch die Variable Threshold des CNHD-Chunks angegeben wird.

Im Header jedes FRDT- (FrameData-) Chunks befinden sich die Variablen Name und Sample. Soll eine CineData-Datei vom WaveTracer DS in Sampledaten konvertiert werden, so ist es zwingend erforderlich, das die Variable "Sample" den kompletten Pfad und den Namen eines IFF-Mono-Samples enthält.

Es ist allerdings nicht notwendig, diesen Pfad schon in dem Programm festzulegen, das die CineData-Datei erzeugt. Hier kann die Variable

```
"Sample" leer bleiben. Der User muß dann natürlich den CineTracer®
benutzen, um die Sample-Pfade anzugeben. Da der CineTracer viele Features
zur Manipulation der CineData-Files bietet, stellt das eigentlich keinen
Umweg dar. Vielmehr sollte das der reguläre Weg sein.

Der Header:

   type IFF_Header=record
      IFF_ID          :string[4];     ='FORM' IFF-Kennung
      Filelength      :long;          = Dateilänge minus 8
      IFF_Type        :string[4];     ='CINM' {CINeMa-data}
   end;

Der CiNedata-HeaDer-Chunk enthält einleitende Definitionen

   type IFF_Chunk=record
      Chunk_ID          :string[4];    = 'CNHD'
      Chunk_Length      :long;

      Frames            :long;         = Anzahl der Frames, für die Rich-
                                         tungsinformationen gespeichert
                                         sind
      FPS               :byte;         = Frames pro Sekunde - die Abspiel-
                                         geschwindigkeit der Animation,
                                         wenn die Anzeigedauer für jedes
                                         Frame gleich ist. Ist FPS=0 so muß
                                         die Variable AnimJiffies benutzt
                                         werden
      pad1              :byte          = Füllbyte;
      Threshold         :integer;      = Schwellwert für die räumliche Ent-
                                         fernung, bei der ein Objekt die
                                         Hörschwelle überschreitet; dieser
                                         Wert darf nicht größer als 15000
                                         sein
      AnimJiffies       :long;         = Gesamtdauer der Animation (in
                                         Jiffies = 1/60 Sekunde); ist dieser
                                         Wert <>0 so haben die einzelnen Frames
                                         unterschiedliche Anzeigezeiten, die
                                         aus den Variablen RelTime bzw.
                                         StartTime und EndTime hervorgehen.
                                         Ist AnimJiffies=0 so muß der Wert FPS
                                         benutzt werden
      reserved1         :long;
   end;

Der FRameDaTa-Chunk enthält die Daten der zu vertonenden Objekte

   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='FRDT'
      Chunk_Length      :long;
      {Es folgt ein kurzer Framedaten-Header}
      Flags             :word;         = Flags, wie mit dem Sample zu ver-
                                         fahren ist:
                                         OFLAG_SOUND=2 - Objekt vertonen
                                         OFLAG_SOUNDEND=8 - Objekt über zeitli-
                                            ches Limit hinaus bis zum Ende des
                                            Samples vertonen
```

```
      Name              :string[20];   = der Name des Objekts;
      Sample            :string[200];  = der Pfad des Samples, mit dem das
                                         Objekt vertont werden soll; dieser
                                         String kann durchaus leer sein, dann
                                         muß der Samplepfad im CineData-
                                         Lademodul festgelegt werden
      {Der Rest des Chunks ist mit den Framedaten gefüllt, die folgende
       Datenstruktur wiederholt sich dabei sooft, bis die Summe aus allen
       Steps dieser Struktur gleich der Anzahl Frames ist}
      x,y,z             :integer;      = die Koordinaten des Objekts zum je-
                                         weiligen Zeitpunkt; hat das Objekt
                                         Koordinaten mit Werten kleiner oder
                                         gleich -32000, so existiert es zu
                                         diesem Zeitpunkt noch nicht und kann
                                         entsprechend nicht vertont werden
      Steps             :word;         = die Anzahl der Frames, bei denen diese
                                         Koordinaten gleich bleiben
      Res2              :integer;      = reserviert
      Volume            :byte          = absolute Lautstärke des Objektes beim
                                         aktuellen Frame in Prozent, es sind
                                         hier Werte im Bereich von 0% bis
                                         200% möglich
      Res1              :byte;         = reserviert
      RelTime           :long;         = gibt in der Einheit "Jiffies" (=
                                         1/60 Sekunde) an, wie lange das
                                         entsprechende Frame der Animation
                                         angezeigt wird (der Wert entspricht
                                         der gleichnamigen Variablen im
                                         ANHD-Chunk von IFF-ANIM´s)
   end;
```

Der TIMEpattern-Chunk enthält Objekte, die nur als Time-Patterns in das CineData-File eingehen:

```
   type IFF_Chunk=record
      Chunk_ID          :string[4];    ='TIME'
      Chunk_Length      :long;
      {Es folgt ein kurzer Framedaten-Header}
      Flags             :word;         = Flags, wie mit dem Sample zu ver-
                                         fahren ist:
                                         OFLAG_TIMEPATTERN=1 - Objekt in die
                                            Timepattern-Liste aufnehmen
      Name              :string[20];   = der Name des Objekts;
      {Anschließend wiederholet sich die folgende Struktur sooft, wie das
       Objekt in der Szene erscheint}
      StartTime         :long;         = Zeitpunkt des Erscheinens des Objektes
                                         in der Einheit Jiffies (= 1/60 Sek.)
      EndTime           :long;         = Zeitpunkt des Verschwindens des Ob-
                                         jektes, ebenfalls in der Einheit
                                         Jiffies
      Res1              :long;         = reserviert
```

Der Chunk

ANIM

beinhaltet den Namen der Animation, die verwendet wurde, um das CineData-File zu synchronisieren. Wurde die Synchronisations-

funktion nicht genutzt, weil die Animation z.B. eine statische Playrate hat, ist dieser Chunk nicht vorhanden.

1.13 dolby

```
  Dolby-Surround®-Sounds vom WaveTracer
  -------------------------------------
```

Der WaveTracer ist seit der Version 1.8 in der Lage, Dolby-Surround®-Sounds abzuspielen, zu verarbeiten und natürlich zu speichern. Hier kann in zwei Qualitätsstufen gespeichert werden:

SQ-Modus - Ist ein ganz normales Stereosample, welches nur über die Amiga®-Kanäle A und B (bzw. D und C) abgespielt werden muß. Dieses Sample enthält alle nötigen (Dolby®-) Informationen, die Qualität ist hier jedoch nicht besonders hoch.

XQ-Modus - Hier werden alle vier möglichen Kanäle verwendet. Die Reihenfolge beim Abspeichern ist hier zwar - wie oben beschrieben - L, R, C und SL, muß aber, da Dolby®-codierte Informationen und keine wirkliche Aufteilung in getrennte Kanäle mehr vorliegt, beim Abspielen auf einem Amiga® auf die Kanäle A, B, C und D gelegt werden ("L" nach A, "R" nach B, "C" nach C und "SL" nach D).

Sollen diese Samples korrekt abgespielt werden, müssen die einzelnen Kanäle beim Amiga® ganz exakt zur gleichen Zeit (und mit der gleichen Samplerate und Lautstärke) abgespielt werden, da der Dolby®-Decoder das Ergebnis sonst nicht akzeptiert und den Sound als normalen Stereo-Sound wiedergibt. Diese Gleichzeitigkeit aller Kanäle ist meines Erachtens nach nur über die direkte Programmierung der Audio-Hardware möglich.

1.14 3byte

```
  Die 3Byte-Kompression
  ---------------------
```

Die 3Byte-Kompression ist eine Kompressionsmethode, die ab dem WaveTracer DS Version Mark II existiert und ausschließlich bei den 24SX Dateien Verwendung findet. Diese Kompressionsmethode wird ohne Einflußmöglichkeit durch den User ständig benutzt. Da aber von früheren WaveTracer-Versionen noch Dateien existieren, bei denen diese Kompressionsmethode noch nicht angewandt wurde, darf nicht einfach davon ausgegangen werden, das alle 24SX-Files in 3Byte komprimiert wurden.

Die 3Byte-Kompression ist eigentlich nichts weiter als eine reine Runtime-Kompression. Die 24Bit-Daten des WaveTracer sind vorzeichenbehaftete 32Bit-Zahlen, deren Bits folgendermaßen genutzt werden:

```
Bit  0 bis Bit 22 - Datenbits
Bit 23 bis Bit 30 - frei
Bit 31            - Vorzeichenbit
```

Daraus folgt, das die 24Bit-Daten sich im Bereich von ca. -8388600 bis 8388600 (-MAX24..MAX24) bewegen.

Dieses Kompressionsverfahren ist ein verlustfreies Verfahren, da die Daten wieder zu 100% hergestellt werden.

Die Kompression:
Die 24Bit-Daten werden durch Addition einfach um 8388600 (+ MAX24) angehoben. Jetzt sind, da das Bit 31 nicht mehr als Vorzeichenbit genutzt wird, die oberen 8 Bit frei. Folglich brauchen nur noch die Bits 0 bis 23, d.h. wirklich nur die unteren 24 Bit, abgespeichert werden.

Die Dekompression:
Hier wird einfach alles in umgekehrter Reihenfolge erledigt: Die 24Bit werden in die unteren 3 Bytes eines Longwords geladen und durch Subtraktion von MAX24 wieder in das korrekte Longword-Format für den WaveTracer umgewandelt.

1.15 delta1

Die Delta-1-Kompression

Die Delta-1-Kompression basiert auf der Grundidee der Soundkompression des MPEG®-Kompressionsverfahrens, welches für bewegte Bilder mit Sound entwickelt wurde.
Beim Delta-1-Kompressionsverfahren - welches sich nur bei Samples mit mehr als einem Kanal verwenden läßt - wird der erste Kanal in vollen 24 Bit (bei 24Bit-Daten standardmäßig 3Byte-komprimiert) abgespeichert und alle folgenden Kanäle in 8 Bit. Der Kniff bei dieser Kompressionsmethode besteht darin, das die Informationen in den in 8 Bit gespeicherten Kanälen sich auf den ersten Kanal beziehen: Es wird der Unterschied, d.h. die Differenz, zum ersten Kanal ermittelt und in nur 8 Bit Tiefe abgespeichert.

Die Delta-1-Kompression sollte nur angewandt werden, wenn sich die verschiedenen Kanäle nicht zu sehr unterscheiden. Bei DTS®/AC-3®-Effekten, bei denen der Sound quer durch den Raum fetzt, ist sie also weniger zu gebrauchen. Aus diesem Grunde wird nach dem Abspeichern auch angezeigt, welche Qualität das Sample noch hat - hier muß dann der User entscheiden, ob ihm das genügt oder nicht.

Dieses Kompressionsverfahren zählt - wie FibonacciDelta - zu den verlustbehafteten Verfahren.

Die Kompression:
Der erste von mehreren Kanälen wird ganz normal abgespeichert (bei 24Bit-Daten wird an diesem Kanal eine 3Byte-Kompression vorgenommen).
Alle weiteren Kanäle beziehen sich immer auf den ersten Kanal:
1. Es wird die Differenz zwischen Kanal 1 und dem aktuellen Kanal ermittelt
2. Aus dem Absolutwert der Differenz wird die Wurzel gezogen und durch 32,26 dividiert (SQRT(ABS(Differenz)) / 32.26)
3. Das Ergebnis, welches sich im Wertebereich einer 7Bit-Zahl bewegt (0..127) wird mit negativem Vorzeichen versehen, wenn die o.g. Differenz kleiner 0 ist

Das Ergebnis dieser Prozedur ist eine vorzeichenbehaftete 8Bit-Zahl, die jedoch nicht weiter mit verlustbehafteten Verfahren wie z.B.

```
   FibonacciDelta komprimiert werden sollte.

Die Dekompression:
   Hier erfolgt wieder die Umkehrung des oben beschriebenen Verfahrens.
   1. Multiplikation der 8Bit-Zahl mit 32,26 und Quadrierung des Ergeb-
      nisses ( SQR(Data8 * 32.26) )
   2. Multiplikation mit -1, wenn der 8Bit-Wert (Variablentyp Short)
      kleiner als 0 war
   3. Berechnung des 24Bit-Wertes des Kanals mittels 24Bit-Wert des
      ersten Kanales und des oben berechneten Differenzwertes

Nach mehreren Versuchen mit dem Delta-1-Kompressionsverfahren mußte ich
feststellen, das eine Differenz mit quadratischer Umrechnung auf 8Bit die
besten Ergebnisse erzielte. Warum hier nicht - wie eigentlich zu erwarten -
eine logarithmische Berechnung die klanglich besseren Ergebnisse bringt ist
mir noch nicht ganz klar.
```

1.16 allgemeines

```
                  Allgemeines zur Programmierung
  ------------------------------

Hier werden einige technische Informationen vermittelt, die für die Pro-
grammierung von neuen Modulen von grundsätzlicher Bedeutung sind. Es wird
dabei auch auf Konstanten eingegangen, die in der Datei WTIncl.mod defi-
niert sind (z.B. MAX24,PlayL24,...).
  Der WaveTracer DS verarbeitet die Sounddaten in 24 Bit. Effektiv kann
aber der komplette 32Bit-Raum von LongWords (vorzeichenbehaftet) benutzt
werden. Wird bei der Bearbeitung eines Sounds seine Amplitude im gesamten
Bereich angehoben (bis max 2^32) oder abgesenkt, so wird er bei der an-
schließenden Optimierung durch den WaveTracer DS wieder in den 24Bit-Be-
reich von -MAX24 bis +MAX24 skaliert.
  Die Anfangsadressen der Datenbereiche befinden sich in Arrays der Größe
[1..6]. Dabei gilt - mit Ausnahme der Adressen MemA16[x] - folgende Zu-
ordnung:   1 - L -   linker Kanal
           2 - R -   rechter Kanal
           3 - C -   Centerkanal
           4 - SL -  bei Quadro/DolbySurround: Kanal hinten
                     bei AC-3: Kanal hinten links
           5 - SR -  Kanal hinten rechts
           6 - Sub - Subwooferkanal
Die verwendeten Speicherbereiche besitzten unterschiedliche Längen und
Verwendungszwecke. Hier gelten folgende Zuordnungen:

Speicherbe-  | Verwendung      | Mit Daten mo- | Maximallänge | benutzter
reich [be-   |                 | mentan beleg- |              | Variablen-
nutzte Ar-   |                 | ter Bereich   |              | typ
rayvari-     |                 |               |              | (Zahlenraum)
ablen        |                 |               |              |
-------------+-----------------+---------------+--------------+--------------
-------------+-----------------+---------------+--------------+--------------
MemA24[x]    | Speicher für zu | PlayL24       | MemL24 (*)   | Long
[1..6]       | bearbeitende    |               |              | (-2^31..2^31)
             | 24Bit-Daten     |               |              |
```

MemAUNDO[x] [1..6]	UNDO-Speicher für 24Bit-Daten	PlayL24	MemL24 (*)	Long (-2^31..2^31)
MemA16[x] [1..2]	Speicher für Playdaten, wird nur vom Wave-Tracer benutzt	PlayL24 / 2	MemL24 / 2	Word (0..65535)
MemAAlpha[x] [1..6]	Alpha-Kanal für zugehörigen 24-Bit-Kanal	PlayL24 / 80	MemL24 / 80	Byte (0..255)
AlphaUNDO[x] [1..6]	UNDO-Speicher für Alpha-Kanal	PlayL24 / 80	MemL24 / 80	Byte (0..255)

Bei den mit (*) markierten Längen handelt es sich um die für Daten verwendbare Maximallänge, die nicht mit der vom WaveTracer DS allocierten Speicherlänge übereinstimmt. Um Sound- bzw. UNDO-Speicherbereiche zu (de-)allocieren, sind die

Funktionen

WTM_ALLOCUNDO, WTM_FREEUNDO,
WTM_ALLOCCHANNEL und WTM_FREECHANNEL zu verwenden!

Die Länge PlayL24 spezifiziert die Länge des reservierten Speicherbereiches, die wirklich mit zu nutzenden Sampledaten belegt ist. Dieser kann nämlich durchaus kleiner sein, als der allocierte Speicherbereich. Wird ein Sample aus irgendeinem Grund länger, so ist PlayL24 entsprechend zu vergrößern. PlayL24 darf natürlich niemals größer als MemL24 werden.

Beim Alpha-Kanal steht immer ein Byte für 20 24Bit-Samples bzw. 20 Longwords. Demzufolge ist dieser immer nur 1/80stel so lang wie der 24Bit-Speicher. Wird der Alpha-Kanal benutzt, ist also immer zu beachten, das ein Wert für zwanzig aufeinanderfolgende 24Bit-Werte gilt. Der Alpha-Kanal-Wertebereich ist linear. 255 entspricht hier einer Amplitude von 100% des 24Bit-Wertes, 128 dementsprechend nur noch 50,196%.

1.17 module

Programmierung von Modulen

Hier soll nun die Programmierung eigener Module beschrieben werden. Nach einer allgemeinen Beschreibung folgt eine Aufschlüsselung auf die verschiedenen Modularten. Es wird dabei auf die speziellen Eigenschaften dieser Module eingegangen. Nach einer Beschreibung aller derzeit verfügbaren WaveTracer-Funktionen, die von den Modulen genutzt werden können, wird zum Schluß noch das für die Programmierung notwendige WaveTracer-Includefile genau beschrieben.

Um die Programmierung etwas anschaulicher zu machen, habe ich mich entschlossen, einige

Sourcecodes

mitzuliefern. Diese wurden alle in
PASCAL geschrieben. Von ihnen sollte aber ohne große Probleme auf andere

Programmiersprachen geschlossen werden können.
Diese Demoprogramme befinden sich im Unterverzeichnis "Sources".

Das Modulehandling erfolgt über ein Messagesystem, das nicht ganz dem entspricht, was Commodore®/Amiga Technologies®/Amiga International® oder wer auch immer so vorschreiben. Die Abweichungen gegenüber der "korrekten" Programmierung beruhen auf Besonderheiten beim WaveTracer DS und stellen keine wirkliche Regelverletzung dar, da hier - im gegensatz zur "normalen" Programmierung - von bestimmten Grundannahmen ausgegangen werden kann. So ist es z.B. als sicher anzusehen, das der WaveTracer-Port immer so lange existent ist, wie ein Modul arbeitet. Weiterhin verändert der WaveTracer die übermittelten Daten niemals nachträglich.

Die Informationsübermittlung vom WaveTracer zu den Modulen geschieht über eine WTStdMsg-Struktur, die sich direkt an die normale, von Exec benutzte Message-Struktur anschließt. Die WTStdMsg-Struktur braucht nach Erhalt nicht kopiert werden, da die Kommunikation nach einer Art Handshake-Prinzip arbeitet: Nachdem das Modul die Message an seinem Port erhalten hat, hat es auch alle Rechte an der WTStdMsg-Struktur. Der WaveTracer nimmt während dieser Zeit daran keinerlei Modifikationen vor, sondern wartet, das das Modul mit seiner Arbeit fertig wird. Ist der Auftrag, den das Modul in WTStdMsg.Flags bekommen hat erledigt, bzw. will das Modul eine WaveTracer-interne Funktion benutzen, werden die Rechte an der Struktur mit PutMsg() zum WaveTracer zurückgegeben. Solange nun der WaveTracer diese Rechte hat, darf das Modul diese Struktur nicht verändern. Das ist erst wieder möglich, wenn mit GetMsg() eine Message empfangen wurde und die Rechte an der WTStdMsg-Struktur demzufolge wieder beim Modul liegen.

Der Ablauf beim Aufruf eines Moduls sieht folgendermaßen aus:

- das Modul kreiert, sofort nachdem es gestartet wurde, einen Port mit einem Namen der Form "WT...Port" (Die Namen sind im Include-File für jeden Modultyp festgelegt
- das Modul wartet darauf, das an diesem Port eine Message eingeht
- der Pointer auf den Port, dem geantwortet werden soll wird aus Message.mn_ReplyPort ermittelt (ACHTUNG! Es kann nicht davon ausgegangen werden, das immer der WaveTracer mit dem "WaveTracerPort" das Modul aufruft!)
- es wird überprüft ob die in WTStdMsg.Version stehende Versionsnummer mit der Konstanten VERSION_... identisch ist. Wenn nicht, wird die Meldung MDE_WRONG_MODULEVERSION in WTStdMsg.Flags eingetragen und die Message an den WaveTracer gesandt. Danach wird das Modul beendet. Wird die Modulversion nicht mit der Versionsnummer der WTStdMsg-Struktur verglichen, könnte die dem Modul bekannte WTStdMsg-Struktur von der vom WaveTracer übermittelten abweichen und dem Modul nicht verständliche Daten beinhalten.
- in der der Message-Struktur folgenden WTStdMsg-Struktur gibt der WaveTracer mittels der Variablen WTStdMsg.Flags ein Kommando, was das Modul zu tun hat: sich identifizieren (MDC_ASKREADY), Parameter definieren (MDC_DEFINEIT), ...
- hat das Modul seine Arbeit getan, wird in der Variablen WTStdMsg.Flags eine Meldung hinterlassen, ob die Operation erfolgreich war, z.B.: Operation ausgeführt (MDE_READY), Operation wurde abgebrochen (MDE_CANCELLED), Datei konnte nicht geöffnet/gefunden werden (MDE_FILE-ERROR),...
- nach Absendung der Meldung in der WTStdMsg-Struktur mittels PutMsg() und dem anschließenden Eintreffen (GetMsg()) einer beliebigen, inhaltlich jetzt irrelevanten Message, beendet das Modul sich selbst

Der WaveTracer stellt verschiedene Funktionen zur Verfügung, die die Programmierung von Modulen vereinfachen und eine einheitliche Benutzerführung sicherstellen sollen.

Die WaveTracer-Funktionen werden durch Flags im Feld PRC_Flags ausgewählt. Je nach der Art der gewählten Funktion werden noch weitere Werte in den PRC_-Variablen benötigt. Schließlich muß die Variable WTStdMsg.Flags noch auf 0 gesetzt werden, da der dort möglicherweise noch vorhandene Wert vom WaveTracer DS als Meldung zum Modulende fehlinterpretiert werden könnte. WaveTracer-Funktionen dürfen also nur dann aufgerufen werden, wenn WTStdMsg.Flags gleich Null ist. Der Aufruf einer WaveTracer-Funktion geschieht folgendermaßen:

- WTStdMsg.Flags auf 0 setzen
- PRC_Flags mit dem gewünschten Flag beschreiben, z.B.: WTM_FILEREQ - der Filerequester soll benutzt werden, WTM_CHANNELREQ - der Kanalrequester soll aufgerufen werden, WTM_TASKMSG - ein Text wird am Bildschirm dargestellt, ...
- Message mit PutMsg() senden und auf Antwort warten
- mittels WaitMsg() und GetMsg() auf eine Message warten; die daran angefügte WTStdMsg-Struktur bestätigt die Ausführung der Funktion und liefert mitunter in den PRC_-Variablen Ergebniswerte

Es kann sein, das Soft- oder Effektmodule (eventuell auch Lademodule) verschiedene Daten vom User abfragen und erhalten müssen. Sofern in der Struktur WTStdMsg.ActWaveOp Variablen frei sind, die dafür genutzt werden könnten, so wird das weiter unten angegeben (diese Struktur darf NICHT von Speichermodulen benutzt werden!). Sind alle belegt oder wird mehr Platz benötigt, als in diesen Variablen vorhanden ist, kann ein zusätzlicher Datenbereich vom Modul allociert und benutzt werden. Dieser Bereich geht automatisch mit in das WTA-Script ein, wird abgespeichert, beim Einladen wieder allociert und mit den gespeicherten Daten belegt sowie vom WaveTracer wieder korrekt freigegeben, wenn er nicht mehr benötigt wird. Die Benutzung erfolgt folgendermaßen:

- ist der Wert von WTStdMsg.MemAData=0, so wurde noch kein Speicher reserviert; es ist, um einen Bereich für zusätzliche Daten festzulegen folgendermaßen zu verfahren:
  - einen ausreichend großen Speicherbereich allocieren
  - die Startadresse dieses Bereiches in die Variable WTStdMsg.ActWaveOp.MemAData schreiben
  - die Größe dieses Speicherbereiches in die Variable WTStdMsg.ActWaveOp.MemLData eintragen
  - die Chunklänge WTStdMsg.ActWaveOp.ChunkSize auf WOP_BIG setzen (Damit wird dem WaveTracer mitgeteilt, das die WTA-Script-Struktur WTStdMsg.ActWaveOp in voller Länge (=WOP_BIG) abgespeichert werden soll und das sich in WTStdMsg.ActWaveOp.MemAData ein Pointer auf einen Datenbereich der Länge WTStdMsg.ActWaveOp.MemLData befindet, der mit zum WTA-Script gehört
- der (bereits zuvor) angelegte Speicherbereich, dessen Startadresse sich in WTStdMsg.ActWaveOp.MemAData befindet kann vom Modul frei benutzt/beschrieben werden

Die Module bearbeiten prinzipiell nur die Bereiche, die durch WTStdMsg.MemA24[x] adressiert werden, da nur sie die relevanten 24Bit-Daten beinhalten. Die Bereiche WTStdMsg.MemA16[x] werden für das Abspielen des Samples benötigt und vom WaveTracer nach jeder ausgeführten Operation neu mit 16Bit-Daten beschrieben. Diese werden aus den 24Bit-Daten aller Kanäle errechnet.

```
**  GRUNDSÄTZLICH sind bei der Programmierung von Modulen, die Daten    **
**  im Speicher des Computers verändern, folgendes zu beachten: Beim    **
**  WaveTracer können im WTA-Script die Daten von Teilen des WTA-       **
**  Scriptes fast beliebig verändert werden. So kann es durchaus sein,  **
**  das bei der anschließenden Neuberechnung des Samples zuvor reser-   **
**  vierte und evtl. auch schon einmal bearbeitete Kanal-Speicherbe-    **
**  reiche nicht mehr existieren! Es ist also - auch wenn die Variab-   **
**  len WTStdMsg.ActWaveOp.Channels, WTStdMsg.ActiveChannels oder       **
**  WTStdMsg.ActiveMode scheinbar gesicherte Daten beinhalten - immer   **
**  zu überprüfen, ob die Anfangsadresse WTStdMsg.MemA24[x] des zu      **
**  bearbeitenden Kanals überhaupt ungleich 0 ist und damit auf einen   **
**  reservierten Speicherbereich zeigt!                                 **
```

1.18 lademodule

```
                  Die Lademodule & das Brainfile »MagicWords.data«
  ------------------------------------------------

Verzeichnis: "LOADER"

Diese Module sind als reine Lademodule für das Einladen von Daten von
einem Speichermedium zuständig. Es sind hier drei Kommandos vom WaveTracer
an das Modul möglich:

MDC_ASKREADY: Es wurde im Funktionsgenerator-Definitionsfenster ein neues
   Lademodul ausgewählt. Dieses soll sich nun durch die Angabe von Typ und
   Copyright identifizieren. Wenn ein Lademodul das MDC_ASKREADY-Kommando
   erhält, wird normalerweise die WTM_TASKREQ-Funktion aufgerufen und
   danach das Modul mit der Meldung MDE_READY verlassen.
     Der WaveTracer hält bei Abgabe dieses Kommandos keinerlei Resourcen
   bereit, da das Modul hier wirklich nichts anderes zu tun hat, als
   den Requester mit den Modul- und Copyrightinformationen aufzurufen.

MDC_DEFINEIT: Der User hat eine Datei selektiert, die in einem späteren
   Schritt vom aktuellen Lademodul eingeladen werden soll. Mit diesem
   Kommando soll diese Sample-Datei erstmalig untersucht werden. Die Datei
   wird dabei bereits vom WaveTracer mit DosOpen() geöffnet. Das für die
   Dos-Operationen benötigte Filehandle vom Typ BPTR befindet sich in
   WTStdMsg.ActFHandle. Die Datei darf vom Modul auch nicht mit DosClose()
   geschlossen werden, das erledigt wiederum der WaveTracer.
     Der WaveTracer erwartet vom Lademodul folgende Daten in folgenden
   Variablen:
   - die Länge des Samples - WTStdMsg.ActWaveOp^.Operator[1]
   - die vorhandenen Kanäle mittels "CH_ "-Flags - WTStdMsg.ActiveChannels
   - der Soundmode ("MD_ "-Konstanten) - WTStdMsg.ActiveMode
   Wird das Format der Datei nicht erkannt (Lademodul und Dateiformat
   passen z.B. nicht zusammen) wird mit der Meldung MDE_NOTMYFORMAT
   beendet. Der WaveTracer gibt dann selbständig einen entsprechenden
   Hinweis an der User.
     Ist alles korrekt verlaufen, wird die Meldung MDE_READY abgegeben und
   das Modul beendet.
     Hier erwartet der WaveTracer, das nur die ActWaveOp-Struktur sowie die
   Variablen WTStdMsg.ActiveMode und WTStdMsg.ActiveChannels mit Daten
```

belegt werden. Die Speicherbereiche MemL24[x] für später einzuladende Sampledaten sind hier noch nicht reserviert worden.

MDC_DOIT: Der User hat das Definitionsfenster des Funktionsgenerators mit dem "OK"-Gadget verlassen. Das Modul wird durch dieses Kommando aufgefordert, das Sample einzuladen.

Das FileHandle der bereits vom WaveTracer mit DosOpen() geöffneten Datei findet sich wieder in der BPTR-Variablen WTStdMsg.ActFHandle.

Das Sample ist mit folgenden Parametern - die der User seit dem Aufruf mit MDC_DEFINEIT durchaus verändert haben könnte (!!) - einzuladen:

- WTStdMsg.ActWaveOp.Operator[1] - die Länge des Samples
- WTStdMsg.ActiveChannels - die einzuladenden Kanäle (dieser Wert kann von den wirklich allocierten Bereichen für die 24Bit-Kanaldaten abweichen)

Die in WTStdMsg.ActWaveOp.Operator[2] gespeicherte Anzahl von Perioden braucht nicht beachtet zu werden. Sollte hier ein Wert >1 stehen, so kopiert der WaveTracer das Sample entsprechend oft.

Enthält die Sampledatei nur einen Kanal, so sind folgende Unterscheidungen zu treffen:

- der Soundmode ist MD_MONO - das Sample ist in den Center-Kanal d.h. an die Adresse WTStdMsg.MemA24[3] zu laden
- ein anderer Soundmode außer MD_MONO wurde eingestellt - das Sample ist in den linken Kanal d.h. an die Adresse WTStdMsg.MemA24[1] zu laden

Das einzuladende Sample muß vom Lademodul in Longwords, also in vorzeichenbehaftete 32Bit-Zahlen umgewandelt werden (Zahlenbereich -MAX24..MAX24) und ist dann - je nach vorhandenen und einzuladenden Kanälen - in die bereits reservierten Speicherbereiche mit den Adressen WTStdMsg.MemA24[1] bis WTStdMsg.MemA24[6] zu legen. Die Größe der reservierten Bereiche steht in der Variablen WTStdMsg.MemL24.

Als Rückgabewert erwartet der WaveTracer die Größe der tatsächlich in die Longwords eingeladenen 24Bit-Sampledaten in der Variablen WTStdMsg.PlayL24 in der Maßeinheit Bytes.

Ein Beispiel: Es soll ein 5000 Bytes langes 8Bit-Sample geladen werden. Der WaveTracer hat für die 24Bit-Daten 22000 Bytes Speicher reserviert. Diese Länge findet sich in WTStdMsg.MemL24. Das Modul lädt nun seine 5000 Bytes Sample ein und wandelt diese in 24 Bit um. Die effektiv vorhandenen 20000 Bytes Sampledaten stehen nun am Anfang des bei WTStdMsg.MemL24[x] beginnenden Speicherbereiches. Um dem WaveTracer nun mitzuteilen, wie Lang das Sample wirklich ist, wird in WTStdMsg.PlayL24 also 20000 eingetragen.

Ist alles korrekt abgelaufen wird wieder die Meldung MDE_READY abgegeben und das Modul beendet.

Bevor der WaveTracer dieses Kommando gibt, wurden natürlich die Speicherbereiche für die über die WTStdMsg.ActiveMode spezifizierten Kanäle allociert und deren Startadresse in den Variablen WTStdMsg.MemA24[x] eingetragen. Auch haben jetzt alle Daten in der WTStdMsg-Struktur Gültigkeit.

ACHTUNG! Die Datei darf niemals mit DosClose(WTStdMsg.ActFHandle) geschlossen werden, das erledigt der WaveTracer!

Ermittlung des Soundmodes

Bei den meisten Soundformaten sind keine Informationen über den Soundmode enthalten. Bei diesen Dateien muß aus der Art und Anzahl der vorhandenen Kanäle auf den Mode geschlossen werden. Dafür gibt es seit dem WaveTracer Mark III die Funktion

WTM_GETSOUNDMODE

.

Das Brainfile »MagicWords.data«

Diese Datei enthält Informationen über die speziellen Kennzeichen der verschiedenen Soundformate und das dazugehörige Lademodul in Form eines ASCII-Textes. "MagicWords.data" wird vom WaveTracer DS immer dann gelesen, wenn der Versuch, ein File mit einem bestimmten Lademodul einzuladen fehlgeschlagen ist und das Lademodul die Fehlermeldung MDE_NOTMYFORMAT zurückgegeben hat.

Im Brainfile gehören immer zwei aufeinanderfolgende Zeilen zusammen. Kommentare und Leerzeilen sowie Leerzeichen sind nicht erlaubt! Wird anstelle von normalen Textzeichen ein "#" angegeben, so wird diese Stelle beim Vergleich mit der zu untersuchenden Datei nicht beachtet und übersprungen. Die Syntax soll im folgenden an zwei Beispielen erläutert werden:

```
FORM####8SVX
UNIVERSAL_IFF
```

Der WaveTracer interpretiert diese beiden Zeilen folgendermaßen: Wenn die zu untersuchende Datei als erstes die Zeichenkette "FORM" und ab der Position 9 (die Positionen 5 bis 8 wurden ignoriert!) die Zeichenkette "8SVX" aufweist, so handelt es sich um eine Datei, die mit dem Lademodul "UNIVERSAL_IFF" geladen werden kann.

```
Creative Voice File
VOICE
```

Beginnt die untersuchte Datei mit dem String "Creative Voice File", so kann sie mit dem Lademodul "VOICE" im "LOADER"-Verzeichnis geladen werden.

Anmerkungen: Weist das Brainfile auf ein Lademodul hin, das sich nicht im Verzeichnis "LOADER" finden läßt, so gibt der WaveTracer eine Fehlermeldung aus.

Wird das Modul gefunden, so werden automatisch die Kommandos MDC_ASKREADY und danach MDC_DEFINEIT an das Lademodul gegeben.

1.19 softmodule

Die Softmodule

Verzeichnis "LOADER"

Softmodule kennen - da sie von der Bedienung her den Lademodulen ähneln - die gleichen drei Kommandos, reagieren auf sie aber teilweise anders:

MDC_ASKREADY: Die Reaktion auf dieses Kommando weicht von den Lademodulen ab. Hier werden, sofern das überhaupt notwendig ist, alle für das Modul wichtigen Daten definiert. Die Variablen WTStdMsg.Operator[4] bis Operator[6] können dabei für eigene Zwecke frei belegt werden.
Ist das geschehen, wird das Modul mit MDE_NICE_SOFTMOD verlassen. Diese Meldung entspricht dem sonst üblichen MDE_READY, muß hier stattdessen aber unbedingt verwendet werden, um dem WaveTracer mitzuteilen, das es sich hier nicht um ein Lademodul handelt.
Sollen keine Informationen vom User benötigt werden, wird - wie bei den Lademodulen - nur ein Requester mit Modul- und Copyrightinformationen aufgerufen und das Modul dann mit MDE_NICE_SOFTMOD (ebenfalls nicht mit MDE_READY!) beendet werden.
Der WaveTracer stellt hier keine besonderen Speicherbereiche zur verfügung. Es können hier lediglich Daten in die WTStdMsg.ActWaveOp-Struktur und evtl. auch in einen an diese Struktur angehängten eigenen

Speicher

eingetragen werden.

MDC_DEFINEIT: Versucht der User irrtümlich, mit einem unter "Lademodul" definierten Softmodul ein Sample auszuwählen, so wird dieses Kommando gegeben. Das Modul darf hier nichts weiter tun, als sofort die Meldung MDE_NICE_SOFTMOD zurückzugeben und sich zu beenden (prinzipiell wären hier noch verschiedene Beschimpfungen des Users möglich, wovon ich aber doch abraten möchte). Der WaveTracer teilt daraufhin dem User selbst mit, das dieses Modul nicht in der Lage ist, Samples einzuladen.

MDC_DOIT: Der User will die Wellenform erzeugen lassen und hat deshalb den Funktionsgenerator mit "OK" verlassen. Das Modul wird somit angewiesen, die Grundwellenform zu generieren. Es muß dabei folgende Werte verwenden:
- die Länge einer Periode (ist für die 24Bit-Longword-Länge wieder mit 4 zu multiplizieren) - WTStdMsg.ActWaveOp^.Operator[1]
- die Anzahl der Perioden (sofern sinnvoll einsetztbar) - WTStdMsg.ActWaveOp^.Operator[2]
- der Phasenwinkel (sofern möglich) - WTStdMsg.ActWaveOp^.Operator[3]
- die zu verwendenden Kanäle - WTStdMsg.ActiveChannels
- der aktuelle Soundmode - WTStdMsg.ActiveMode

Die Maximallänge ist wieder in WTStdMsg.MemL24 zu finden. In der Variablen WTStdMsg.PlayL24 ist vom Modul die real mit Sampledaten belegte Länge (32 Bit oder Samples x 4) einzutragen. Wurde alles korrekt ausgeführt, ist - das erste und einzige mal bei Softmodulen - MDE_READY zurückzugeben und das Modul zu beenden.
Der WaveTracer hat hier natürlich wieder alle - wie schon bei den

Lademodulen

beschriebenen - nötigen Speicherbereiche reserviert.

1.20 speichermodule

Die Speichermodule

```
Verzeichnis: "SAVER"

Diese Module sind von allen am einfachsten zu handhaben. Sie kennen nur ein
Kommando: MDC_DOIT. Nach Erhalt dieses Kommandos speichert das Modul die
Daten in einer bereits vom WaveTracer geöffneten Datei. Wie und in welcher
Auflösung sowie welche Datenblöcke zuvor gespeichert werden müssen, hängen
von der Art des Speichermoduls und damit vom unterstützten Format ab.
  Folgende Daten sind erforderlich und werden vom WaveTracer geliefert:
  - die zu speichernden Kanäle - WTStdMsg.ActiveChannels
  - die Länge der zu speichernden Daten - WTStdMsg.PlayL24 (wieder im Long-
    wordformat, es ist also evtl. durch 2 oder 4 zu dividieren)
  - das Filehandle der bereits durch den WaveTracer geöffneten Zieldatei -
    WTStdMsg.ActFHandle
  - die Speicherbereiche mit den zu speichernden Daten - WTStdMsg.MemA24[x]
Wurde erfolgreich abgespeichert, wird sollte mittels der Funktion
WTM_TASKREQ ein Hinweis auf Art und Copyright des Moduls gegeben. Danach
wird die Meldung MDE_READY abgegeben und das Modul beendet.

ACHTUNG! Auch hier darf die Datei nicht mit DosClose(WTStdMsg.ActFHandel)
         geschlossen werden, das erledigt wieder der WaveTracer.
```

1.21 effektmodule

```
  Die Effektmodule
  ----------------

Verzeichnis: "EFFECTS"
besonderes Suffix: " .eff"
zusätzliche Dateien: " .eff.data"

Dieser Modultyp ist der aufwendigste, da hier sehr viele verschiedene
Anwendungsmöglichkeiten bestehen. Die ganze hier mögliche Funktions-
vielfalt verbirgt sich hinter zwei möglichen Kommandos: MDC_DEFINEIT und
MDC_DOIT. Zusätzliche Informationen verbergen sich in der Datei
" .eff.data": die Art und Verwendung des Moduls sowie die Imagedaten für
das Symbol im WTA-Script und Effektmodul-Requester. Die Dateien
" .eff.data" haben folgenden Aufbau:

    Flags (long)  - 0 - keine Definitionsmöglichkeit, die Operation wird
                        sofort ausgeführt (z.B. "DeNoiser.eff", "HQ.eff")
                    1 - EFF_DEFWIN, es müssen zuvor verschiedene Parameter
                        eingestellt werden, der WaveTracer soll also das
                        Kommando MDC_DEFINEIT geben (z.B. "FastEcho.eff",
                        "Tone.eff")
                    2 - EFF_NOCALC, nach korrekter Bearbeitung der Sound-
                        daten durch das Modul soll keine Neugenerierung
                        der Play-Daten erfolgen (z.B. "SetLoop.eff",
                        "SetLoopOff.eff"), da keine Veränderungen an den
                        Sampledaten vorgenommen wurden.
                          Die genannten Beispiel-Effektmodule dürften hier
                        die einzigen Anwendungsmöglichkeiten sein, für die
                        dieses Flag überhaupt in Frage kommt.
    Imagedaten 64 x 24 Punkte, Plane 1
```

```
    Imagedaten 64 x 24 Punkte, Plane 2
    Imagedaten 64 x 42 Punkte, Plane 3
    (Das Image hat zwar eine Größe von 64 x 24, jedoch werden nur 53 x 24
     Punkte ausgenutzt; zum Neuzeichnen bitte die Vorlage "Eff.Data.pic"
     verwenden)

Auf die beiden möglichen Kommanos ist folgendermaßen zu reagieren:
MDC_DEFINEIT: Es ist in der Datei " .eff.data" das Flag EFF_DEFWIN gesetzt
   worden. Demzufolge wird das Modul durch das Kommando aufgefordert, vom
   User alle nötigen Definitionen zu verlangen. Dem Modul steht hierbei
   fast die gesamte Struktur WTStdMsg.ActWaveOp zur Verfügung. Deren frei
   verfügbare Variablen sollten folgendermaßen belegt werden:
     Chunksize - Größe der belegten Teile der ActWaveOp-Struktur; wenn zu-
                 sätzliche Datenbereiche benötigt werden, ist hier WOP_BIG
                 einzutragen, sonst ist vom WaveTracer WOP_NORM voreinge-
                 stellt
     OpType - evtl. vorhandene Unterteilung (im Funktionsgenerator wären
                 das z.B. Zahlenwerte die den verschiedenen möglichen
                 Wellenformen entsprechen
     Channels - Kanäle, die das Modul bearbeiten soll
     Flags - frei verfügbar
     Operator[1]..Operator[6] - frei verfügbar
   Die Programmierung des Definitionsfensters wird durch WaveTracer-
   Funktionen wie z.B. WTM_OPENDWIN (Definitionsfenster öffnen),
   WTM_AUTODWIN (Definitionsfenster nur mit Kanalgadgets), oder
   WTM_LEAVEWIN ("OK"-/"Abbruch"- Gadgets abfragen und Fenster schließen)
   unterstützt.
     Effektmodul-Definitionsfenster sollten normalerweise immer die 6
   Kanalgadgets sowie ein "OK"- und ein "Abbruch"-Gadget besitzen. Wird
   ein Fenster durch den User mit "OK" verlassen, so wird als Meldung
   MDE_READY an den WaveTracer zurückgegeben. Wird hingegen das "Abbruch"-
   Gadget betätigt, ist in WTStdMsg.Flags MDE_CANCELLED einzutragen. Der
   WaveTracer entfernt dann den letzten Schritt (also das abgebrochene
   Effektmodul) wieder aus dem WTA-Script.
     Es ist bei diesen Modulen zu beachten, das sie ebenfalls aus dem
   WTA-Script heraus aufgerufen werden können. Es müssen dann jedoch die
   zuvor vom User getätigten Einstellungen zu sehen sein. Deshalb ist bei
   der Initialisierung der Moduldaten wie folgt zu verfahren:
      - Wird ein Modul zum ersten mal aufgerufen, so ist die Variable
        WTStdMsg.ActWaveOp.Operator[1] gleich -1. Ist das der Fall,
        so können alle im Modul verwendeten Variablen mit den Default-
        werten beschrieben werden und evtl. nötige zusätzliche Speicher-
        bereiche initialisiert werden. Die Variable WTStdMsg.ActWaveOp.
        Operator[1] ist dabei auf einen Wert<>-1 zu setzen.
      - Findet sich in der Variablen WTStdMsg.ActWaveOp.Operator[1] ein
        Zahlenwert<>-1, so ist das Modul bereits aufgerufen worden. Es
        ist also bei einer erneuten Definition von Parametern mit den
        vorhandenen Variablen-Werten und mit dem evtl. bereits reservier-
        ten Speicherbereich zu arbeiten.

MDC_DOIT: Dieses Kommando kann nicht durch Flags in der Datei " .eff.data"
   beeinflußt werden. Es veranlaßt, das das Modul die Sampledaten bearbei-
   tet. Folgende Variablen sind hierbei wichtig:
   - WTStdMsg.ActWaveOp^.Channels - die zu bearbeitenden Kanäle (ist nach
     Abfrage der im Definitionsfenster selektierten Kanäle oder aber durch
     pauschale Einstellung auf alle Kanäle durch das Modul selbst zu be-
     schreiben)
```

- WTStdMsg.MemA24[1] bis MemA24[6] - die Anfangsadressen der Sampledaten
- WTStdMsg.PlayL24 - die genutzte Länge der vorhandenen Daten
- WTStdMsg.MemL24 - die maximal nutzbare Länge
- WTStdMsg.ActWaveOp-Struktur - alle hier vom Modul selbst belegten Variablen

Vor der Bearbeitung ist die Funktion WTM_WORKINFO aufzurufen. Diese zeigt ein Kurzinfo an (normalerweise der Name des Moduls) und stellt die Statusanzeige dar, die anzeigt, wie weit die Arbeit am Sample fortgeschritten ist (0%..100%).

Soll ein Effektmodul in der Lage sein, nur bestimmte, durch den User markierte Teile des Samples zu bearbeiten, so ist hier die WaveTracer-Funktion WTM_GETMARKOFFSET zu verwenden. Diese liefert die Offsets für Anfang und Ende des markierten Bereiches in absoluten Bytes, also direkt auf die 24Bit-Datenbereichsadressen MemL24[x] anwendbaren Werten.

Während der Bearbeitung der Sampledaten ist in Abständen die Funktion WTM_GETABORTINFO aufzurufen. Diese Funktion gibt als Antwort zurück, ob während der Bearbeitung die "Esc"-Taste gedrückt wurde, d.h. ob die Bearbeitung abgebrochen werden soll. Ist das der Fall, ist das Modul mit der Meldung MDE_ERROR zu beenden.

Wurde das Sample hingegen erfolgreich bearbeitet, ist wieder MDE_READY zu verwenden.

Bei der Bearbeitung der Sampledaten muß wiederum beachtet werden, das dieses Kommando vom WTA-Script aus durch die Betätigung des "Ausführen"-Gadgets gegeben worden sein könnte. So kann es z.B. sein, das gar nicht mehr alle in WTStdMsg.ActWaveOp.Channels definierten Kanäle vorhanden sind, weil z.B. der Soundmode verändert worden ist. Es ist also grundsätzlich zu überprüfen, ob der zu den geforderten Kanälen gehörende Speicherbereich überhaupt allociert wurde. Ist die Variable WTStdMsg.MemA24[x] gleich Null, so darf dieser Kanal NICHT mehr bearbeitet werden (wenn das doch jemand versuchen sollte, macht er Bekanntschaft mit dem Typen, der sich bis Kick 1.3 noch "GURU" nannte)!

Wird die effektive Samplelänge bei der Bearbeitung der Sampledaten verändert, so gibt es folgende Bearbeitungsmöglichkeiten:

- Alle Sample-Kanäle werden gleichmäßig kürzer: Hier ist einfach die neue Länge der 24Bit-Daten in WTStdMsg.PlayL24 einzutragen.
- Die einzelen Samplekanäle können kürzer werden, es läßt sich aber nicht vorhersagen, welcher Kanal um wieviel kürzer wird: Hier muß der WaveTracer feststellen, wie lang das Sample jetzt ist. Um das zu ermöglichen, ist der 24Bit-Bereich vom neuen Sampleende des aktuell bearbeiteten Kanals bis zum vorherigen Sampleende zu löschen, d.h. auf 0 zu setzen. Ein Beispiel: Die für die einzelnen Kanäle allocierten Speicherbereiche sind 20000 Bytes lang und das Sample hat eine reale Länge von 19000 Bytes (-> WTStdMsg.PlayL24). Es handelt sich um ein Stereo-Sample, also sind WTStdMsg.MemA24[1] und WTStdMsg.MemA24[2] die Startadressen für die Kanäle CH_LEFT und CH_RIGHT.

  Nach der Bearbeitung des linken Kanals ist dieser nur noch 18500 Bytes lang. Folglich muß der Speicher von der Adresse WTStdMsg.MemA24[1] + 18500 bis nach WTStdMsg.MemA24 + WTStdMsg.PlayL24 gelöscht werden.

  Beim rechten Kanal, der nach der Bearbeitung nur noch 18200 Bytes lang ist, wird genauso verfahren.

  Die Variable WTStdMsg.PlayL24 bleibt unverändert. Nach der anschließenden Beendigung des Moduls mit MDE_READY untersucht der WaveTracer das Sample und legt die neue effektive Samplelänge PlayL24 mit 18500 Bytes fest.

- Das Sample wird gleichmäßig oder unterschiedlich länger, bleibt aber kleiner oder gleich der Maximallänge MemL24: Da der WaveTracer die Speicherbereiche von WTStdMsg.MemA24[x]+PlayL24 bis WTStdMsg.MemA24[x]+MemL24 immer automatisch löscht, braucht bei der Verlängerung nichts weiter beachtet werden. Vor der Beendigung des Samples wird nur im Wert WTStdMsg.PlayL24 die Gesamt-Größe des Speichers WTStdMsg.MemL24 eingetragen. Der WaveTracer stellt dann die reale Länge wieder selbständig fest.
- Das Sample wird größer als der Gesamtspeicher WTStdMsg.MemL24: Diese Variante ist nach Möglichkeit unbedingt zu vermeiden, da sie die komplizierteste und bei versionsbedingten Veränderungen auch die inkompatibelste ist. Hier ist - sollte es sich nicht umgehen lassen - folgendes zu tun:
  Es sind alle in WTStdMsg-Struktur eingetragenen Startadressen WTStdMsg.MemA24[x] (Länge gleich WTStdMsg.MemL24), WTStdMsg.MemAUNDO[x] (Länge gleich WTStdMsg.MemL24), WTStdMsg.MemA16[x] (Länge gleich WTStdMsg.MemL24 div 2), WTStdMsg.MemAAlpha[x] (Länge gleich round(WTStdMsg.MemL24/80+5)) und WTStdMsg.AlphaUNDO[x] (Länge gleich round(WTStdMsg.MemL24/80+5)) zu untersuchen. Sind diese Adressen ungleich 0, so sind sie neu zu allocieren und bei den WTStdMsg.MemA24[x]-Adressen sowie den WTStdMsg.MemAAlpha[x]-Adressen die Daten zu kopieren. Ist das geschehen, können die alten, zu kurzen Speicherbereiche freigegeben werden (dabei NUR die alten Längen MemL24, MemL24 div 2 etc. verwenden!!).
  Eine Ausnahme bilden dabei die Speicherbereiche WTStdMsg.MemA24[x] und WTStdMsg^.MemAUNDO[x]. Diese dürfen NICHT mit AllocMem() oder FreeMem() (de-)allociert werden! Hier sind unbedingt immer die WaveTracer-Funktionen WTM_ALLOCUNDO, WTM_FREEUNDO, WTM_ALLOCCHANNEL und WTM_FREECHANNEL zu verwenden!
  Die neuen Speicherbereiche sollten nach Möglichkeit im FastRAM liegen und MÜSSEN mit dem Flag MEMF_CLEAR allociert werden. Zum Schluß sind natürlich in die Variablen WTStdMsg.MemL24 die neue Gesamtlänge und in WTStdMsg.PlayL24 die neue, effektiv mit Sampledaten belegten Länge einzutragen.
  Die Speicherbereiche WTStdMsg.MemAUNDO[x] sind hier gesondert zu behandeln. Sie sind zweckmäßigerweise ganz zu Anfang komplett freizugeben (WTM_FREEUNDO) und erst am Ende neu zu allocieren (WTM_ALLOCUNDO). Gelingt es nicht, den erforderlichen Speicher zu belegen, weil z.B. nicht mehr genug vorhanden ist, so sind die evtl. bereits allocierten WTStdMsg.MemAUNDO-Kanäle wieder freizugeben und alle MemAUNDO-Startadressen auf 0 zu setzen. Es steht dann ab sofort kein Speicher für die UNDO-Funktion mehr zur Verfügung. Gleiches gilt für die AlphaUndo-Startadressen, sofern diese zuvor schon belegt waren.
  Als zweites ist unbedingt noch - egal ob der UNDO-RAM alloziert werden konnte oder nicht - die Variable WTStdMsg.UndoPossible auf "false" zu setzen. Damit wird dem WaveTracer mitgeteilt, das sich der letzte Schritt nicht mit "UNDO" rückgängig machen läßt, da die dafür nötigen Daten nicht mehr vorhanden sind.

Eine BESONDERHEIT ergibt sich für Module, die die Länge des Samples verändern und/oder mindestens teilweise die Lautstärke verändern. Hier ist zu beachten, das möglicherweise ein Alpha-Kanal verwendet wird, dessen Länge dann ebenfalls verändert werden muß.
Bei Längenveränderungen sind die Daten in den Alpha-Kanälen entsprechend zu verlängern oder zu kürzen. Die sich neu ergebenden Werte sind nicht

einfach duch Verdopplung oder Auslassungen zu ermitteln, sondern duch eine möglichst genaue Interpolation.
  Veränderungen der Lautstärke sind nur dann direkt an den Sampledaten vorzunehmen, wenn kein Alpha-Kanal vorhanden ist. Ist das jedoch der Fall, so ist der Verlauf der Alphakanal-Daten entsprechend der gewünschten Veränderungen zu modifizieren.

1.22 wavetracer-funktionen

```
                  Die WaveTracer-Funktionen
  -------------------------
```

Der WaveTracer stellt verschiedene Funktionen bereit, die von den Modulen genutzt werden können. Damit wird erreicht, das der Programmcode der Module kürzer und die Programmierung einfacher wird und das der WaveTracer auch bei Modulen von verschiedenen Autoren ein einheitliches Bedienkonzept und Erscheinungsbild bietet.
  Für den Aufruf einer WaveTracer-Funktion muß zuerst WTStdMsg.Flags auf 0 gesetzt werden. Danach wird in PRC_Flags einer der unten beschriebenen Werte eingetragen um die Art der aufzurufenden Funktion festzulegen. In allen weiteren PRC_ -Varibalen müssen dann noch weitere Werte eingetragen werden, deren Sinn und Verwendung im folgenden genau beschrieben werden soll.
  Wird nach Absendung (mit PutMsg()) und erneutem Empfang (mit GetMsg()) der Message in WTStdMsg.PRC_Long1 der Wert -1 vorgefunden, so konnte die gewünschte Funktion nicht korrekt ausgeführt werden oder es wurde eine Funktion aufgerufen, die in der benutzten Version des WaveTracer nicht existiert. Der Rückgabewert -1 ist für alle fehlgeschlagenen Funktionen einheitlich, auch wenn in der folgenden Beschreibung keine Rückgabewerte angegeben sind.
  Alle beschriebenen Variablen befinden sich in der WTStdMsg-Struktur. Deshalb wird zur Vereinfachung der sonst zu verwendende Präfix "WTStdMsg.WTMsgPrc." weggelassen.

```
WTM_TASKREQ=$0001
   Funktion: Ein Requester mit  maximal zwei Textzeilen und mit ein bis
      drei möglichen Gadgets wird geöffnet
   Variablen: PRC_Str1 - die erste Textzeile
              PRC_Str2 - die zweite Textzeile
              PRC_Str3 - das linke Gadget
              PRC_Str4 - das mittlere Gadget
              PRC_Str5 - das rechte Gadget
      Wird in den Variablen für die Gadgetnamen NIL oder '' eingegeben,
      so wird das zugehörige Gadget nicht mit dargestellt. Jeder
      Taskrequester muß aber mindestens ein Gadget besitzen (standard-
      mäßig das mittlere Gadget als "OK"-Gadget)
   Rückgabewert: PRC_Long1 gibt an, welches Gadget betätigt wurde:
      1 - Links (=Esc), 2 - Mitte (=Space), 3 - Rechts (=Return)

 WTM_FILEREQ=$0002
   Funktion: Mittels eines Filerequesters (der "reqtools.library") kann
      eine Datei ausgewählt werden
   Variablen: PRC_Str1 - kompletter Zielpfad; Enthäld dieser String einen
                         Pfad, so werden Verzeichnis und Filename in
```

```
                    PRC_Str2 und PRC_Str3 ignoriert
            PRC_Str2 - Zielverzeichnis, falls bekannt
            PRC_Str3 - Default-Filename, falls vorhanden
            PRC_Str4 - Dateimuster der anzuzeigenden Dateien nach der
                       amtlichen AMIGA®-Syntax (z.B. "#?.eff" - nur Da-
                       teien mit der Endung " .eff" werden im Filere-
                       quester angezeigt), wird hier NIL oder '' einge-
                       tragen, werden alle Dateien mit Ausnahme von
                       Dateien, die auf ".p", ".backup" oder ".o"
                       enden, angezeigt.
            PRC_Str5 - Fenstertitel des Filerequesters
                       (z.B. "Datei LADEN")
   Rückgabewerte: PRC_Str1 - ausgewählter Pfad
                  PRC_Str2 - ausgewähltes Verzeichnis
                  PRC_Str3 - ausgewählter Filename

WTM_GETABORTINFO=$0003
   Funktion: Dem Modul wird mitgeteilt, ob der User während der Bearbei-
      tung eines Samples die "Esc"-Taste oder das "Abbruch"-Gadget gedrückt
      hat
   Variablen: keine
   Rückgabewerte: PRC_Long1 gibt an ob abgebrochen werden soll
       0 - "Esc"/"Abbruch" nicht betätigt, fortsetzen
      -1 - "Esc"/"Abbruch" wurde betätigt, Die Operation ist sofort
           abzubrechen und das Modul mit MDE_ERROR zu beenden

WTM_TASKMSG=$0004
   Funktion: Ein kurzer Text wird in einem gerahmten Feld im oberen
      Bildschirmdrittel ausgegeben (wie z.B. bei "Oktave 1","Oktave 2",...)
   Variablen: PRC_Str1 - der auszugebende Text
   Rückgabewerte: keine
Diese Funktion darf nicht verwendet werden, wenn zuvor WTM_WORKINFO einge-
setzt wurde. Der Text würde dann verdeckt werden.

WTM_WORKINFO=$0005
   Funktion: Ein kurzer Text wird ausgegeben und das "0%..100%"-Statusfenster
      wird geöffnet
   Variablen: PRC_Str1 - der auszugebende Text
   Rückgabewerte: keine
Diese Funktion ist unbedingt am Anfang jedes Effektmoduls auszuführen, um
das Statusfenster zu öffnen. Sollen während der weiteren Bearbeitung Infor-
mationen ausgegeben werden, so sollte dazu ebenfalls diese Funktion ge-
nutzt werden. WTM_TASKMSG verbietet sich hier, da deren Text vom Status-
fenster verdeckt werden würde.

WTM_HANDLEMOD=$0006
   Funktion: Ein anderes Modul wird aufgerufen; die Kommandos an das neu
             aufgerufene Modul werden dabei über eine zweite WTStdMsg-
             Struktur in der Variablen WTStdMsg.Flags gegeben
   Variablen: PRC_Str1 - Portname des aufzurufenden Moduls (darf nicht
                         mit dem Namen des Ports des aufrufenden Moduls
                         übereinstimmen, im Zweifelsfall ist der Name des
                         eigenen Ports zu ändern), z.B. "WTLoaderPort"
              PRC_Str2 - kompletter Pfad des aufzurufenden Moduls, z.B.
                         "EFFECTS/Scale.eff"
              PRC_Long1 - Startadresse der zu sendenden Message mit
                          einer angefügten und entsprechend ausgefüllten
```

zweiten WTStdMsg-Struktur.
Die WTStdMsg-Struktur muß alle Daten enthalten, die das Modul für die Ausführung eines Kommandos benötigt. Welche Kommandos bei welchen Modultypen zu geben sind, wurde für

Lade-,

Soft-,

Effekt-
und
Speichermodule

bereits ausführlich beschrieben.
Das Modul, das die Funktion WTM_HANDLEMOD aufruft, übernimmt dabei die Arbeit des WaveTracers, was die Bereitstellung von Ressourcen angeht. Wann wo welcher Speicher bereitgestellt werden muß, wurde bei oben genannten Modul-Beschreibungen bereits erläutert.

Rückgabewerte: PRC_Str1 - "0" bei erfolgreicher Ausführung; die eigentlich wichtigen und nötigen Rückgabewerte befinden sich natürlich in der WTStdMsg-Struktur. Es kann hier erforderlich sein, einen vom zweiten Modul für zusätzliche Daten allocierten Speicher wieder freizugeben. Das geschieht mit FreeMem(WTStdMsg.ActWaveOp^.MemAData,WTStdMsg.ActWaveOp.MemLData).
Um WaveTracer-Funktionen, die das aufgerufenen Modul benutzen will, braucht man sich nicht kümmern. Die entsprechenden Aufrufe werden vom WaveTracer abgefangen und bearbeitet. Das aufrufende Modul erhält wirklich nur die Rückgabewerte bei Beendigung des aufgerufenen Moduls.

WTM_AUTODWIN=$0007
Funktion: Automatisches Definitionsfenster, in dem nur die zu verwendenden Kanäle bestimmt werden können
Variablen: PRC_Str1 - Titel des Definitionsfensters
PRC_Long1 - Kanäle des aktuellen Soundmodes, die bereits angewählt werden sollen, normalerweise wird hier der Wert WTStdMsg.UsedChannels verwendet
Rückgabewerte: PRC_Long1 - vom User ausgewählte Kanäle

WTM_OPENDWIN=$0008
Funktion: Es wird ein Standard-Definitionsfenster mit Gadgets geöffnet; die Gadgetstrukturen werden vom WaveTracer verwaltet, deshalb muß dieses Fenster dann auch mit WTM_LEAVEWIN geschlossen werden
Variablen: PRC_Str1 - Titel des Fensters
PRC_Long1 - Höhe des Fensters
PRC_Long2 - Offset vom oberen Bildschirmrand; wird 0 angegeben, wird das Fenster am unteren Bildschirmrand geöffnet
PRC_Long3 - Flags für die Art der vom WaveTracer anzulegenden Gadgets; diese haben die GadgetID 1 bis 8, die dann für weitere Gadgets natürlich nicht mehr verwendet werden dürfen.
Gadgetflags: 1 - "OK"-/"Abbruch"-Gadgets werden angelegt; hierbei erhält das "OK"Gadget die ID 1 und das

```
                                        "Abbruch"-Gadget die 2
                                    2 - Kanalgadgets werden angelegt;
                                        diese haben die GadgetID 3
                                        bis 8
              PRC_Long4 - Anzahl der Selectlisten, die über den Kanal-
                          gadgets plaziert werden sollen. Selectlisten
                          können mit der Funktion WTM_DOSELECTLIST verwendet
                          werden.
              PRC_NewPtr - muß auf NIL gesetzt werden
   Rückgabewerte: PRC_NewPtr - Pointer auf die Window-Struktur

WTM_LEAVEWIN=$000A
   Funktion: Das Definitionsfenster, das zuvor mit WTM_OPENDWIN geöffnet
             wurde, wird wieder geschlossen
   Variablen: PRC_Long1 - RawKey-Code aus der IntuiMsg, der die Beendigung
                          veranlaßt hat (RAWKEY von der "Return"- oder der
                          "Esc"-Taste)
              PRC_Long2 - GadgetID aus der IntuiMsg, der die Beendigung
                          veranlaßt hat (1 oder 2 von "OK"- oder "Abbruch"-
                          Gadget)
              PRC_NewPtr - Zeiger auf die Window-Struktur des Definitions-
                          fensters
   Rückgabewerte: PRC_Long1 - Art der Beendigung:
                  0 - Abbruch (MDE_CANCELLED muß an den WaveTracer gemeldet
                      werden, damit dieser diese Operation wieder aus dem
                      WTA-Script entfernt)
                  1 - normal Fortsetzen (Meldung MDE_READY an den
                      WaveTracer, Effektmodul geht mit in das WTA-Script ein)

WTM_SETCHANNELGADS=$000B
   Funktion: Die Kanalgadgets eines mit WTM_OPENDWIN geöffneten Definiti-
      onsfensters werden, je nach vorhandenen Kanälen, aktiviert und
      selektiert
   Variablen: PRC_Long1 - Aktueller Soundmode, normalerweise also
                          WTStdMsg.ActiveMode
              PRC_Long2 - defaultmäßig anzuwählende Kanalgadgets, norma-
                          lerweise WTStdMsg.UsedChannels
   Rückgabewerte: keine

WTM_GETCHANNELGADS=$000C
   Funktion: Die Kanalgadgets des mit WTM_OPENDWIN geöffneten Fensters
      werden auf Selektierung hin untersucht
   Variablen: keine
   Rückgabewerte: PRC_Long1 - durch die Kanalgadgets gewählte Kanäle

WTM_ALLOCALPHA=$000D
   Funktion: Allociert für alle vorhandenen Kanäle Speicherbereiche für
      den Alpha-Channel
   Variablen: keine
   Rückgabewerte: bei Mißerfolg PRC_Long1=-1 ansonsten sind die Start-
      adressen der Alpha-Kanäle in den Variablen WTStdMsg^.AlphaMemA zu
      finden

WTM_GETMARKOFFSET=$000E
   Funktion: Liefert die Offsets für Anfang und Ende des zu bearbeitenden
      Bereiches eines Samples für die Effektmodule, die auch in der Lage
      sind, Sampleteile zu bearbeiten.
```

```
        Diese Offsets sind das Äquivalent des im Soundeditor markierten
      Bereichs eines Samples. Wurde kein Bereich markiert, umfassen die
      Offsets das gesamte Sample. "Offset Anfang" ist dann gleich 0 und
      "Offset Ende" gleich PlayL24.
   Variablen: keine
   Rückgabewerte: PRC_Long1 - Offset Anfang
                  PRC_Long2 - Offset Ende
      Um aus den Offsets absolute Adressen zu bekommen, sind diese mit den
      Startadressen des jeweils zu bearbeitenden Kanals zu addieren.
        Soll ein Alphakanal bearbeitet werden, so sind diese Offsets durch
      80 (= 4 x 20) zu dividieren und dann mit den Alphakanal-Startadressen
      zu addieren.

WTM_RESTOREALPHA=$000F;
   Funktion: Setzt den Alphakanal auf den Amplitudenverlauf des Samples um
      und bringt alle Alphakanal-Daten wieder auf den Maximalwert 255.
   Variablen: PRC_Long1 - Flags der zu bearbeitenden Kanäle,
                          z.B. WTStdMsg.UsedChannels
              PRC_Long2 - Start-Offset des zu bearbeitenden 24Bit-Bereiches
              PRC_Long3 - End-Offset des zu bearbeitenden 24Bit-Bereiches
              Die Werte für Start- und End-Offset können mit Hilfe der
              Funktion WTM_GETMARKOFFSET ermittelt werden.
   Rückgabewerte: keine

WTM_PREFSPROCESS=$0010
   Funktion: Die 24Bit- und 16Bit-Sampledaten werden neu bearbeitet
   Variablen: PRC_Long1 - Flags der anzuwendenden Bearbeitungsstufen
                          $1 - Generierung des Subwoofers, Optimierung der
                               Samplelänge, evtl. Verdopplung der Sample-
                               länge, Optimierung der Gesamtlautstärke;
                               alle Operationen auf 24Bit-Ebene
                          $2 - Berechnung der 16Bit-Daten
                          $4 - frei
                          $8 - Zeichnen der Wellenformen aus den 24Bit-Daten
                         $10 - Unaufgefordertes Abspielen des kompletten
                               Sounds, wenn der WaveTracer DS entsprechend
                               konfiguriert wurde
   Rückgabewerte: keine

WTM_ALLOCUNDO=$0011;
   Funktion: Allociert Speicher für 24Bit- und evtl. auch für Alpha-UNDO-
      Daten. Hierbei ist vom Modul zu prüfen, ob bereits Speicher allociert
      wurde, da diese Funktion voraussetzt, das keiner vorhanden ist.
   Variablen: keine
   Rückgabewerte: Neue Anfangsadressen in den MemAUNDO[x]-Variablen, wenn
      Speicher allociert werden konnte.

WTM_FREEUNDO=$0012;
   Funktion: Gibt den Speicher aller belegten 24Bit- und Alpha-UNDODaten
      frei.
        Diese Funktion muß unbedingt benutzt werden, bevor mit WTM_ALLOC-
      CHANNEL oder WTM_FREECHANNEL Veränderungen bei den Soundspeicher-
      bereichen vorgenommen werden. Wurden alle Veränderungen ausgeführt,
      kann versucht werden, mit WTM_ALLOCUNDO wieder UNDO-Speicher zu
      allocieren.
   Variablen: keine
   Rückgabewerte: Die Anfangsadressen aller UNDO-Speicherbereiche in der
```

```
      WTStdMsg-Struktur werden auf 0 gesetzt.

WTM_FREEPLAYLIST=$0013;
   Funktion: Gibt den Speicher der durch die Time-Pattern- und
      Playlist-Entry-Strukturen belegt wurde frei und löscht damit die
      aktuellen Time-Pattern- und Playlisten vollständig
   Variablen und Rückgabewerte: keine

WTM_ADDPATTERN=$0014;
   Funktion: Fügt an das Ende der Time-Pattern-Liste eine leere
      PlayListPattern-Struktur an (die zwar anders heißt, aber eine Time-
      Pattern-Struktur ist), die vom Modul mit Daten gefüllt werden muß.
   Variablen: keine
   Rückgabewerte: PRC_NewPtr - Zeiger auf die leere PlayListPattern-Struktur

WTM_ADDENTRY=$0015;
   Funktion: Fügt hinter einem vorgegebenen Listeneintrag in der Playlist-
      Entry-Liste eine leere PlayListEntry-Struktur ein, die ebenfalls
      ihre Daten vom Modul erhalten muß.
   Variablen: PRC_Long1 - Position in der PlayListEntry-Liste, hinter der
      die neue Struktur eingefügt werden soll; $0FFFFFFF wenn die neue
      Struktur am Ende der Liste angehängt werden soll
   Rückgabewerte: PRC_NewPtr - Zeiger auf die leere PlayListEntry-Struktur

WTM_GETCHANNELBITS=$0016;
   Funktion: Gibt die zu einem Soundmode gehörenden Kanäle an
   Variablen: PRC_Long1 - Soundmode
   Rückgabewerte: PRC_Long1 - alle zu diesem Soundmode gehörenden Kanäle

WTM_GETSOUNDMODE=$0017;
   Funktion: Ermittelt den Soundmode aus den vorhandenen Kanälen
   Variablen: PRC_Long1 - vorhandene Kanäle
   Rückgabewerte: PRC_Long1 - der zugehörige Soundmode

WTM_DOSELECTLIST=$0018;
   Funktion: Stellt eine Selectliste dar
   Variablen: PRC_NewPtr - Pointer auf eine ausgefüllte
                SelectListGad
                -
      Struktur
   Rückgabewerte: PRC_Long1 - Nummer des ausgewählten Selectlisteneintrags

WTM_LONGREQ=$0019;
   Funktion: Requester, der vom User die Eingabe eines Zahlenwertes fordert
   Variablen: PRC_Str1 - Requestertext
              PRC_Str2 - Text des linken Requestergadgets
              PRC_Str3 - Text des mittleren Gadgets
              PRC_Str4 - Text des rechten Requestergadgets
              PRC_Long1 - Default-Zahlenwert, der im Integergadget stehen
                          soll
              PRC_Long2 - Minimal möglicher Zahlenwert
              PRC_Long3 - Maximal erlaubter Zahlenwert
   Rückgabewerte: PRC_Long1 - Nummer des betätigten Gadgets, 1 - Links,
                            2 - Mitte, 3 - Rechts
                  PRC_Long2 - vom User eingegebener Zahlenwert

WTM_DRAWTIMEPATTERNLIST=$001A;
```

```
   Funktion: Die Liste der Timepatterns, die im "Time-Patterns"-Fenster
             dargestellt ist, wird aktualisiert. Diese Funktion ist
             immer dann auszuführen, wenn (Time-)Patterns (=Daten der
             PlayListPattern-Struktur) hinzugefügt oder entfernt wurden.
             Die Abfrage, ob das Fenster überhaupt geöffnet ist, wird
             dabei vom WaveTracer DS ausgeführt.
   Variablen: keine
   Rückgabewerte: keine

WTM_ADDANIMPATH=$001B;
   Funktion: Es wird der Pfad zu einer Animation übergeben. Diese Animation
             wird dann, abhängig von der Nummer eines eventuell selektierten
             TimePatterns, im "Anim-Frames"-Fenster dargestellt, sofern
             dieses geöffnet ist.
   Variablen: PRC_Str1 - Der komplette Pfad zur Animation
   Rückgabewerte: keine

WTM_ALLOCCHANNEL=$001C;
   Funktion: alloziert einen einzelnen (noch nicht vorhandenen!) Kanal für
             Sounddaten. Es ist zu beachten, das der UNDO-Speicher davon
             nicht beeinflußt wird und deshalb ggf. zuvor mit WTM_FREEUNDO
             freigegeben werden muß. Wurde der Kanal alloziert, kann mit
             WTM_ALLOCUNDO wieder UNDO-Speicher angefordert werden.
   Variablen: PRC_Long1 - Länge des zu reservierenden Bereiches (normaler-
                          weise also WTStdMsg^.MemL24)
   Rückgabewerte: PRC_Long1 - Anfangsadresse des neuen Kanals; dieser muß,
             wenn er dem WaveTracer zur Verfügung stehen soll, in die ent-
             sprechende Variable WTStdMsg^.MemA24 eingetragen werden.
   ACHTUNG: Es sollte bei Speicherbereichen, die für Sounds verwendet
             werden, IMMER diese Funktion benutzt werden. Der kürzere Weg
             über AllocMem() funktioniert NICHT, da der WaveTracer bei
             Sound-Speicherbereichen ein spezielles Management nutzt.
               Weiterhin ist zu beachten, das bei Veränderungen an der
             Anzahl der Kanäle auch der Soundmode verändert wird (Vari-
             ablen WTStdMsg^.ActiveMode und WTStdMsg^.ActiveChannels).

WTM_FREECHANNEL=$001D;
   Funktion: deallociert einen einzelnen Kanal für Sounddaten. Es ist zu
             beachten, das der UNDO-Speicher davon nicht beeinflußt wird
             und deshalb zuvor mit WTM_FREEUNDO freigegeben werden muß.
             Wurde der Kanal alloziert, kann mit WTM_ALLOCUNDO wieder
             UNDO-Speicher angefordert werden.
   Variablen: PRC_Long1 - Anfangsadresse des Kanals
             PRC_Long2 - Länge des Speicherbereiches (normalerweise also
                         WTStdMsg^.MemL24)
   Rückgabewerte: keine; wenn dem WaveTracer der deallocierte Kanal zur
             Verfügung gestanden hat, muß die entsprechende Variable
             WTStdMsg^.MemA24 anschließend auf 0 gesetzt werden, um dem
             Programm mitzuteilen, das der Kanal nicht mehr vorhanden ist.
   ACHTUNG: Es sollte bei Speicherbereichen, die für Sounds verwendet
             werden, IMMER diese Funktion benutzt werden. Der kürzere Weg
             über FreeMem() funktioniert NICHT, da der WaveTracer bei
             Sound-Speicherbereichen ein spezielles Management nutzt.
               Weiterhin ist zu beachten, das bei Veränderungen an der
             Anzahl der Kanäle auch der Soundmode verändert wird (Vari-
             ablen WTStdMsg^.ActiveMode und WTStdMsg^.ActiveChannels).
```

```
WTM_ALLOC_SPECIAL_DATA=$001E;
   Funktion: allociert einen Speicherbereich für eine leere SpecialData-
             Struktur des angegebenen Typs in der richtigen Größe
   Variablen: PRC_Long1 - SD_-Flag für den Typ der zu allocierenden SD-
                          Struktur
   Rückgabewerte: PRC_NewPtr - Pointer auf die leere SD-Struktur

WTM_PLAYKEYS=$0020
   Funktion: Ausführung der auf Tastendruck wählbaren Funktion der
      WaveTracer-eigenen Tastaturbelegung
   Variablen: PRC_Long1 - RawKey-Code der gedrückten Taste
              PRC_Long2 - Playdauer in Millisekunden, wird hier 0 angegeben,
                          wird das komplette Sample abgespielt
              PRC_NewPtr - Zeiger auf die Window-Struktur des Fensters, in
                          dem die Taste gedrückt wurde und in dem eine
                          weitere Tastenbetätigung z.B. zum Abbruch des
                          Sounds erwartet wird
   Rückgabewerte: keine

WTM_SPECIAL_DATA_FOUND=$0040
   Funktion: Das (Lade-)Modul teilt dem WaveTracer mit dieser Funktion mit,
      das besondere Daten (außer den Sampledaten) gefunden wurden.
   Variablen: PRC_Long1 - SD_-Flag, z.B. SD_ALPHA_CHANNEL, wenn eine Datei
                          einen Alphakanal enthält
              PRC_NewPtr - Pointer auf den Speicherbereich des SDBody,
                          sofern Daten gefunden wurden, für die es einen
                          SDBody gibt; der WaveTracer entscheidet dann
                          selbständig, ob er diese SpecialData nutzt.
                            Der SDBody-Speicherbereich darf durch das
                          Modul NICHT freigegeben werden, das wird vom
                          WaveTracer DS zu gegebener Zeit erledigt.
   Rückgabewerte: keine
   Hinweis: Werden in einer Sampledatei Alphakanal-Daten gefunden, so MUSS
      das dem WaveTracer mit dieser Funktion mitgeteilt werden, damit er
      entsprechend Speicher reserviert. Daraus kann aber nicht geschlossen
      werden, das dieser Speicher auch wirklich reserviert wurde. Es könnte
      durchaus passieren, das der User nur die reinen Sampledaten nutzen
      möchte, und deshalb den vorhandenen Alphakanal gar nicht einladen
      will!
        Die Funktion WTM_SPECIAL_DATA_FOUND wird bei Lademodulen zweimal
      benutzt: wenn die Kommandos MDC_DEFINEIT und MDC_DOIT gegeben wurden.
      Anschließend an MDC_DOIT sind - sofern Speicher reserviert wurde -
      die Alphakanal-Daten einzuladen.

WTM_PLAY=$0080
   Funktion: Spielt das aktuelle Sample ab
   Variablen: PRC_Long1  - Playratete; wird 0 eingesetzt, wird in der einge-
                           stellten Playrate abgespielt
              PRC_Long2  - Playdauer in Millisekunden; wird hier 0 einge-
                           setzt, wird das gesamte Sample abgespielt
              PRC_NewPtr - aktuelles Window, das von der Playroutine auf
                           Tastaturbetätigungen überwacht werden soll.
                           Hier wird normalerweise ein Zeiger auf das
                           WaveTracer-Window WtStdMsg^.WTWindow oder aber
                           auf das Definitionsfenster eingetragen.
   Rückgabewerte: keine
```

```
WTM_CHANNELREQ=$0200
   Funktion: Kanalrequester, erfragt Kanäle, die benutzt werden sollen
   Variablen: PRC_Long1 - Vorhandene Kanäle, aus denen ausgewählt werden
                          können soll
              PRC_Long2 - Kanäle, deren Kanalgadgets bereits selektiert
                          werden sollen
              PRC_Long3 - Kompressionsmethode, die benutzt werden soll
                          (COMPRESSION_FIBONACCI_DELTA, COMPRESSION_
                           DELTA oder COMPRESSION_DELTA_2); zweckmäßiger-
                          weise nur bei Speicheroperationen einzusetzen
              PRC_Str1 - Text, der im Kanalrequester stehen soll (z.B.
                         "Bitte Kanäle auswählen!")
   Rückgabewerte: PRC_Long1 - ausgewählte Kanäle
                  PRC_Long2 - ausgewählte Kompressionsmethode

WTM_GETNOTEFREQ=$0800
   Funktion: Notenrequester, aus dem ein bestimmter Notenwert ausgewählt
      werden kann
   Variablen: keine
   Rückgabewerte: PRC_Long1 - Frequenz der ausgewählten Note
   Hinweis: In der
                BlueNote-Struktur
                 wird zusätzlich eingetragen, welche
            Note ausgewählt wurde. Mit dieser Information lassen sich dann
            Notenwert, -name und -playrate ermitteln.

WTM_CALCWAVE=$1000
   Funktion: Die 16Bit Daten eines Samples werden erneut berechnet; diese
             Funktion tut das gleiche wie WTM_PREFSPROCESS mit Flag=2 und
             sollte deshalb zukünftig nicht mehr verwendet werden.
   Variablen: keine
   Rückgabewerte: keine

WTM_WORK_BACKGROUND=$80000000
   Diese spezielle Variable stellt keine Funktion dar, sondern ein Flag,
   das zusammen mit einer WTM_-Funktionsvariablen dem WaveTracer anzeigt,
   das das Modul im Hintergrund weiterarbeitet. In diesem Fall ist das
   Messagehandling natürlich nach den korrekten Reference-Manual-Vorgaben
   zu programmieren, da die sonst gültigen WaveTracer-typischen Erleich-
   terungen bzw. Einschränkungen dann nicht mehr gültig sind.
     Diese Spezial-Funktion ist zwar vorbereitet aber in der aktuellen
   Version noch nicht nutzbar. Da ein Modul, das unbedingt im Hinter-
   grund weiterlaufen muß die absolute Ausnahme darstellen sollte, wird
   die Funktion dieses Flags nur nach Rücksprache mit dem Autor aktiviert.
```

1.23 include

```
                  Das WaveTracer-Includefile
  -------------------------

Im folgenden soll die Bedeutung und die Verwendung der Daten im File
"WTincl.mod" erklärt werden.
```

```
{$if not def WTINCL_MOD}
CONST WTINCL_MOD=0;
```

Die Konstanten vom Typ MDC_ sind alle Kommandos, die vom WaveTracer bei der Ausführung von Modulen an diese gegeben werden. Diese Konstanten werden vom WaveTracer bzw. von einem Modul, das mit der WaveTracer-Funktion WTM_HANDLEMOD ein zweites aufruft, in WTStdMsg.Flags eingetragen.

```
CONST             {*** MDC_ MoDuleCommands ***}
 MDC_ASKREADY=1;   - das Modul soll sich mit Info's und Copyright identi-
                     fizieren (nicht infizieren!!)
 MDC_DEFINEIT=2;   - notwendige Definitionen sollen vorgenommen werden
 MDC_DOIT=8;       - das Modul soll eine Operation ausführen
```

Nachdem der WaveTracer ein Kommando an das Modul gegeben hat, versucht dieses, die geforderte Operation auszuführen. Ob und mit welchem Erfolg das geschehen ist, meldet das Modul mit den MDE_-Konstanten in WTStdMsg.Flags zurück, bevor es selber endet.

```
CONST             {*** MDE_ MoDuleErrors ***}
 MDE_READY=1;                - Operation erfolgreich ausgeführt
 MDE_CANCELLED=2;            - Definition(-sfenster) wurde vom User abge-
                               brochen, dieser Bedienschritt soll nicht
                               mit ins WTA-Script übernommen werden
 MDE_FILEERROR=3;            - Datei hat sich nicht (oder nicht vollstän-
                               dig) bearbeiten lassen; ein Dateifehler/
                               DOS-Error ist aufgetreten; die zugehörige
                               Fehlermeldung wird vom WaveTacer ausgegeben
 MDE_ERROR=4;                - sonstige Fehler bzw. die Operation wurde vom
                               User abgebrochen (siehe
                WTM_GETABORTINFO
                )
 MDE_NOTMYFORMAT=8;          - Datei besitzt ein dem Lademodul unbekanntes
                               Format
 MDE_WRONG_MODULEVERSION=16; - WTStdMsg.Version und Modulversion sind
                               nicht identisch und damit nicht kompatibel,
                               der WaveTracer gibt daraufhin selbständig
                               wieder die entsprechende Fehlermeldung aus
 MDE_NO_MEMORY=32;           - nicht genug Speicher vorhanden; die Fehler-
                               meldung kommt wiederum vom WaveTracer
 MDE_BREAK=64;               - nichtöffentliche Konstante
 MDE_CHANNELERROR=128;       - das Lademodul konnte keine Übereinstimmungen
                               zwischen den in der Sounddatei vorhandenen
                               und vom WaveTracer geforderten Kanälen fest-
                               stellen
 MDE_NICE_SOFTMOD=512;       - Softmodul-Operation erfolgreich ausgeführt
                               oder Modul gibt sich auf das MDC_DEFINEIT-
                               Kommando hin als Softmodul zu erkennen;
                               diese Meldung entspricht dem MDE_READY, ist
                               aber bei Softmodulen stattdessen zurückzu-
                               geben
```

Module können Funktionen nutzen, die bereits im WaveTracer implementiert

```
sind. Zu diesem Zweck wird WTStdMsg.Flags auf 0 gesetzt und die erforder-
liche WTM_-Konstante in PRC_Flags geschrieben. Die genaue
                Beschreibung
                 der
WTM_-Konstanten ist bereits weiter oben erfolgt.

CONST             {*** WTM_ WaveTracerMessage ***}
 WTM_TASKREQ             =$0001;
 WTM_FILEREQ             =$0002;
 WTM_GETABORTINFO        =$0003;
 WTM_TASKMSG             =$0004;
 WTM_WORKINFO            =$0005;
 WTM_HANDLEMOD           =$0006;
 WTM_AUTODWIN            =$0007;
 WTM_OPENDWIN            =$0008;
 WTM_LEAVEWIN            =$000A;
 WTM_SETCHANNELGADS      =$000B;
 WTM_GETCHANNELGADS      =$000C;
 WTM_ALLOCALPHA          =$000D;
 WTM_GETMARKOFFSET       =$000E;
 WTM_RESTOREALPHA        =$000F;
 WTM_PREFSPROCESS        =$0010;
 WTM_ALLOCUNDO           =$0011;
 WTM_FREEUNDO            =$0012;
 WTM_FREEPLAYLIST        =$0013;
 WTM_ADDPATTERN          =$0014;
 WTM_ADDENTRY            =$0015;
 WTM_GETCHANNELBITS      =$0016;
 WTM_GETSOUNDMODE        =$0017;
 WTM_DOSELECTLIST        =$0018;
 WTM_LONGREQ             =$0019;
 WTM_DRAWTIMEPATTERNLIST=$001A;
 WTM_ADDANIMPATH         =$001B;
 WTM_ALLOCCHANNEL        =$001C;
 WTM_FREECHANNEL         =$001D;
 WTM_ALLOC_SPECIAL_DATA =$001E;
 WTM_PLAYKEYS            =$0020;
 WTM_SPECIAL_DATA_FOUND =$0040;
 WTM_PLAY                =$0080;
 WTM_CHANNELREQ          =$0200;
 WTM_GETNOTEFREQ         =$0800;
 WTM_CALCWAVE            =$1000;
 WTM_WORK_BACKGROUND=$8000000;


Die CH_-Flags geben in Variablen wie z.B. WTStdMsg.ActiveChannels oder
WTStdMsg.ActWaveOp.Channels darüber Auskunft, welche Kanäle vorhanden
sind, benutzt werden sollen, selektiert wurden etc.

CONST              {*** CH_ CHannel ***}
 CH_LEFT=1;     - linker Kanal
 CH_RIGHT=2;    - rechter Kanal
 CH_CENTER=4;   - Center-Kanal
 CH_SLEFT=8;    - beim Soundmodus MD_SURROUND hintere Kanal, bei MD_AC3_Sub,
                  MD_AC3 und MD_QUADRO der Kanal hinten links
 CH_SRIGHT=16;  - Kanal hinten rechts
 CH_SUB=32;     - Subwoofer-Kanal
```

Die MD_-Konstanten geben in Variablen wie z.B. WTStdMsg.ActiveMode Auskunft über den eingestellten Soundmode. Diese Information ist bei räumlichen Berechnungen wichtig, da hier von der MD_Konstante auf die Lautsprecherkoordinaten in den Variablen WTStdMsg.PosSurround, WTStdMsg.PosQuadro oder WTStdMsg.PosAC3 geschlossen werden kann. Diese Soundmodi unterscheiden sich in der Aufstellung der Lautsprecher. Deren Positionen sind in Koordinaten-Arrays (ChCoords-Record) enthalten, die weiter unten beschrieben werden.

Der Soundmode läßt sich unter Angabe der vorhandenen Kanäle mit der Funktion WTM_GETSOUNDMODE ermitteln.

```
CONST             {*** MD_ MoDe ***}
```

Soundmode	Erklärung / Kanäle	Koordinaten-Record
MD_MONO=1	Mono / C	WTStdMsg.PosSurround
MD_STEREO=2	Stereo / L, R	WTStdMsg.PosSurround
MD_3CH=3	3Channel / L, C, R	WTStdMsg.PosSurround
MD_QUADRO=4	DTS®-Quadro / L, R, SL, SR	WTStdMsg.PosQuadro
MD_SURROUND=5	Dolby-Surround® / L, C, R, SL	WTStdMsg.PosSurround
MD_AC3_Sub=6	DTS®/AC-3® ohne Subwoofer / L, C, R, SL, SR	WTStdMsg.PosAC3
MD_AC3=7	DTS®/AC-3® mit Subwoofer / L, C, R, SL, Sub, SR	WTStdMsg.PosAC3

In den Konstanten PORT_ sind die Namen der Ports der unterschiedlichen Modultypen definiert. Diese werden für die Funktion CreatePort() benötigt, mit der der Moduleport angelegt wird.

```
CONST             {*** Portnames ***}
 PORT_WT='WaveTracerPort';         - WaveTracer-Port, für das Hauptpro-
                                     gramm reserviert
 PORT_EFFECTMOD='WTEffectModPort'; - Effektmodul
 PORT_LOADER='WTLoaderPort';       - Lade- und Softmodul
 PORT_SAVER='WTSaverPort';         - Speichermodul
```

Die VERSION_-Konstanten geben die bei dieser Version des WaveTracer aktuellen Versionsnummern der Module an. Diese müssen mit dem Wert in WTStdMsg.Version verglichen werden. Sind die beiden Werte nicht gleich, so ist dieses Modul unverzüglich mit der Meldung MDE_WRONG_MODULEVERSION zu beenden.

```
CONST             {*** Portversions ***}
 VERSION_WT=1;        - nichtöffentliche Versionsnummer
```

```
 VERSION_EFFECTMOD=4; - Effektmodul
 VERSION_LOADER=2;    - Lade- und Softmodul
 VERSION_SAVER=3;     - Speichermodul
```

Die KOMPRESSION_-Flags geben die Art der Kompression an, mit der ein Sample komprimiert wurde. Diese Flags werden ebenfalls von der Funktion WTM_CHANNELREQ in der Variablen PRC_Long3 akzeptiert.

```
CONST             {*** Samplekompressions ***}
 COMPRESSION_FIBONACCI_DELTA=$1; - FibonacciDelta
 COMPRESSION_3BYTE          =$2; - 3Byte-Runtime
 COMPRESSION_DELTA          =$4; - Delta-1-Kompression
 COMPRESSION_DELTA_2        =$8; - Delta-2-Kompression
```

Die EFF_-Flags werden in Form eines Longwords noch vor den Imagedaten in den " .eff.data"-Dateien zu den Effektmodulen benötigt. Sie sagen aus, wie das Modul vom WaveTracer behandelt werden soll.

```
CONST
 EFF_DEFWIN=$1;     - es muß vor der Bearbeitung der Sampledaten ein Defi-
                      nitionsfenster geöffnet werden, d.h. der WaveTracer
                      soll erst das Kommando MDC_DEFINEIT geben und erst
                      danach MDC_DOIT
 EFF_NOCALC=$2;     - die Sampledaten werden durch das Effektmodul nicht
                      beeinflußt, eine erneute Berechnung der 16Bit-Daten
                      ist also nicht nötig
```

Die Konstante MAX24 definiert die Maximalwerte der 24Bit-Sampledaten (Datenbereich: -MAX24..MAX24). Diese entspricht, wie zu sehen ist, nicht ganz dem Wertebereich einer 24Bit Zahl. Das ist jedoch kein Fehler, sondern ein Kniff, um Rechenungenauigkeiten bei der Sampleamplituden-Optimierung auszugleichen. Bei der Division durch 65536 bzw. durch 256 erhält man - nach korrekter Rundung - Werte, die den 8Bit- bzw. 16Bit-Raum vollständig ausschöpfen.

```
Const
 MAX24=8388600;
```

MAXSIZE definiert die maximal mögliche Samplelänge in der Einheit Samples. Die reale Samplelänge der 24Bit-/Longword-Daten erhält man wieder durch die Multiplikation mit 4.

```
Const MAXSIZE=50000000;
```

Die SpecialData-Typen geben in einem SDHeader an, um welche Art spezieller Daten es sich handelt. Anhand dieser Typen-Nummer ist dann auch ersichtlich, welcher SDBody verwendet werden muß, und was die Daten in diesem Body bedeuten

```
CONST             {*** SpecialData-Typen ***}
 SD_SHORT_TEXT=$1;     - Kurzer Text mit maximal 100 Zeichen
 SD_TEXT=$2;           - nichtöffentliches Flag
 SD_ANIMINFO=$3;       - Animationsdaten
```

```
 SD_IMAGE=$4;          - nichtöffentliches Flag
 SD_LOOP=$5;           - Anzahl der Loops
 SD_MARKER_ARRAY=$8;   - nichtöffentliches Flag
 SD_PLAYLIST=$10;      - Datei enthält Playlisten-Daten
 SD_ALPHA_CHANNEL=$20; - Datei enthält Alphakanal; hierfür existiert keine
                         SDBody-Struktur, dieses Flag teilt dem WaveTracer
                         vielmehr mit, das Speicher reserviert werden muß,
                         wenn der Alphachannel geladen werden soll
 SD_SZENE=$40;         - der aktuelle Sound ist nur ein Teil eines größeren,
                         zusammengehörenden Ganzen

type r_SDHeader=^SDHeader;
type SDHeader=record;
        NextSpecialData :r_SDHeader; - Zeiger auf den nächsten SDHeader,
                         wenn mehrere Special Data vorkommen
        sdh_Size        :long; - Größe des gesamten Speicherbereiches, in
                         dem sich SDHeader und SDBody befinden
        sdh_Type        :long; - SpecialData-Typ (= SD_-Konstante), gibt
                         an, welche Art von Daten vorhanden sind und welche
                         Art von SDBody zu verwenden ist
     end;

type r_SDBodyShortText=^SDBodyShortText;
type SDBodyShortText=record;   {*** ShortText-SpecialData-Typ SD_SHORT_TEXT ***}
        sdb_SDHeader    :SDHeader;
        sdb_TextString  :string[100]; - Textstring für maximal 100 Zeichen (in-
                                 klusive chr(0))
     end;

type r_SDBodyANIM=^SDBodyANIM; {*** ANIM-SpecialData-Typ SD_ANIMINFO ***}
type SDBodyANIM=record
        sdb_SDHeader    :SDHeader;
        sdb_Frames      :long; - Anzahl der Frames der Animation
        sdb_FPS         :byte; - Playrate in Frames pro Sekunde
        sdb_Flags       :byte; - unbenutzt, ist auf 0 zu setzen
        sdb_pad         :word; - Füllbytes
     end;

type r_SDBodySzene=^SDBodySzene {*** Szenen-SpecialData-Typ SD_SZENE ***}
type sd_BodySzene=record
        sdb_SDHeader    :SDHeader;
        sdb_Name        :string[200]; - Szenenname;
        sdb_StartFrame  :long; - erstes Frame der Szene;
        sdb_EndFrame    :long; - letztes Frame der Szene;
     end;

type r_SDBodyLoop=^SDBodyLoop {*** Loop-SpecialData-Typ SD_LOOP
type SDBodyLoop=record
        sdb_SDHeader            :SDHeader;
        sdb_Loop                :word; - Anzahl der Loops
        sdb_pad                 :word; - Füllbytes
     end;

Im BlueNote-Record sind sämtliche z.B. bei einem Notenrequester angezeig-
ten Notenwerte mit Namen, SampleFrequenz, und der entsprechenden Amiga-
Playrate definiert.
  Wurde die Funktion MDE_GETNOTEFREQ aufgerufen, so steht die Array-Posi-
```

```
tion der ausgewählten Note in den Koordinaten bn_PosX und bn_PosY. Will man
also neben der bei dieser Funktion in PRC_Long1 zurückgegebenen Samplefre-
quenz zusätzlich z.B. auch den Notennamen haben, so findet sich dieser in
der Variablen bn_NoteName[bn_PosX,bn_PosY].

type BlueNote=record
        bn_NoteName                         :array[0..6,1..12] of String[4];
        bn_SampleFreq                       :array[0..6,1..12] of real;
        bn_PlayRate                         :array[0..6,1..12] of word;
        bn_PosX,bn_PosY                     :short;
     end;


Das WTA-Script setzt sich aus Records vom Typ WaveOp zusammen. Diese sind
als Array organisiert. Der Funktionsgenerator steht dabei immer an Position
1, alle weiteren Funktionen jeweils eine Position höher. Die Bedeutung und
Verwendung der einzelnen Variablen, die Daten für das Effektmodul enthalten
können, wurde bereits weiter oben erklärt.

type r_WaveOp=^WaveOp;
type WaveOp=Record
        ChunkSize,SoundMode                 :byte;
        Operation,OpType,Channels,Flags     :integer;
        Operator                            :array[1..6] of long;
        WAreaA,WAreaE                       :long;
           { -->  42 Bytes = WOP_SMALL }
        OpString                            :string[81];
           { --> 123 Bytes = WOP_NORM  }
        MemLData,MemAData                   :long;
           { --> 131 Bytes = WOP_BIG   }
     end;


Die folgende Struktur wird benötigt, wenn eine Selectliste benutztwerden
soll. Die Funktion
                WTM_DOSELECTLIST
                 benötigt
die darin enthaltenen Informationen, um die Selectliste zu benutzen.

type ITTArr=array [1..15] of ^IntuiText;
type r_SelectListGad=^SelectListGad;
type SelectListGad=record
        GadWindow       :^Window; - Pointer auf das Window, in dem sich das
                                    Selectlistengadget befindet
        SGad            :^Gadget; - Pointer auf das Selectlistengadget (ein
                                    ganz normales Boolgadget mit dem Select-
                                    listenimage WTImages^.GImg2)
        SelectedGad     :byte; - der momentan selektierte
                                    Punkt der Selectliste, diese Variable
                                    wird durch die Funktion WTM_DOSELECTLIST
                                    automatisch verändert
        pad1,pad2,pad3  :byte; - Füllbytes bzw. für spätere Erweiterungen
                                    vorgesehen
        ITextTag        :ITTArr; - ein Array bestehend aus Pointern auf die
                                    IntuiTexte der Selectlisteneinträge.
                                    Diese Variable ist zwar als Array gestal-
                                    tet, muß aber wie eine Taglist behandelt
```

```
                                    werden, d.h. es können hier beliebig viele
                                    Pointer aufeinander folgen (also auch mehr
                                    als die 15 im Array definierten); die
                                    Taglist muß mit einem Null-Pointer (NIL
                                    oder NULL) beendet werden. Die Anzahl der
                                    Selectlisteneinträge wird von der
                                    Funktion WTM_DOSELECTLIST aus der Länge
                                    der Taglist ermittelt.


const
   WOP_SMALL=42;
   WOP_NORM=123;
   WOP_BIG=131;


In der WT_Images-Struktur befinden sich Zeiger auf sämtliche beim
WaveTracer verwendeten Images. Diese müssen bei eigenen Gadgets ebenfalls
verwendet werden, um ein einheitliches Erscheinungsbild zu gewährleisten.
  Die Größe des Images kann u.U. von der Gadgetgröße abweichen. In diesem
Fall ist dann die reale Imagegröße zusätzlich in Klammern angegeben.

type r_WTImages=^WTImages;
type WTImages=Record
        HiSys      :boolean; - wird nicht mehr verwendet, da der WaveTracer
                               jetzt Kick 3.0 voraussetzt
        Share      :boolean; - Informationen über den Status des
                               WaveTracer; ist Share=true dann läuft die
                               eingeschränkte Demoversion (wird diese
                               Variable von einem Modul auf "false" ge-
                               setzt, ändert das natürlich auch nichts
                               daran).
        XXX        :^Image;  - nicht mehr belegt
        KnobImg    :^Image;  - Regler-Knopf in Standard-Propgadgets,
                               32 x 11 Punkte
        CalcImg    :^Image;  - Calc-Gadget (" x2|:2 "), 32 x 13 Punkte
        GImg1      :^Image;  - einfaches Gadget, 128 x 16 Punkte
        GImg2      :^Image;  - Selectlisten-Gadget, 128 x 16 Punkte
        GImg3      :^Image;  - Diskgadget, nicht selektiert, 45 x 21
                               (64 x 21) Punkte
        GImg4      :^Image;  - Diskgadget, selektiert, 45 x 21 (64 x 21)
                               Punkte
        GImg5      :^Image;  - Acceptgadget, 128 x 16 Punkte
        GImg6      :^Image;  - Sinus-Wellenform, 64 x 21 Punkte
        ButtonImg1 :^Image;  - einfacher, leerer Knopf, 21 x 12 (32 x 12)
                               Punkte
        ButtonImg2 :^Image;  - Knopf mit Häkchen, 21 x 12 (32 x 12) Punkte
        ButtonImg3 :^Image;  - Knopf mit Note, 21 x 12 (32 x 12) Punkte
        None1      :^Image;  - frei
        None2      :^Image;  - frei
        None3      :^Image;  - frei
        Notes      :^BlueNote; - Zeiger auf die komplett ausgefüllte
                               BlueNote-Struktur
        OKIText    :^IntuiText; - bei "OK"-Gadgets zu verwendender Text
        CIText     :^IntuiText; - bei "Abbruch"-Gadgets zu verwendender
                                  Intuition-Text
        ButtonImg4 :^Image;  - Knopf mit Punkt für Radio-Buttons, 21 x 12
```

```
                                (32 x 12) Punkte
        ButtonImg5 :^Image;  - Knopf mit Pfeil nach oben, 21 x 12
                                (32 x 12) Punkte
        ButtonImg6 :^Image;  - Knopf mit Pfeil nach unten, 21 x 12
                                (32 x 12) Punkte
     end;
```

Die MsgPrc-Struktur ist für den Datenaustausch bei der Benutzung der WaveTracer-Funktionen zuständig. Hier werden alle nötigen Werte eingetragen. Die genaue Verwendung der WTM_-Konstanten, die in den PRC_Flags eingetragen werden und die Verwendung aller anderen PRC_-Variablen wurde weiter oben bereits ausführlich beschrieben.

```
type r_MsgPrc=record
        PRC_Flags                                                :long;
        PRC_Str1, PRC_Str2, PRC_Str3, PRC_Str4, PRC_Str5         :string[200]
        PRC_Long1,PRC_Long2,PRC_Long3,PRC_Long4,PRC_Long5        :long;
        PRC_NewPtr                                               :ptr;
     end;
```

In der ChCoords-Struktur sind die Koordinaten der vorhandenen Lautsprecher für die unterschiedlichen Soundmodi eingetragen. Diese sind bei räumlichen Berechnungen unbedingt zu verwenden. Der Center-Lautsprecher (ChX[3], ChY[3]) hat dabei immer die Position 0,0. Alle anderen Positionen beziehen sich dann auf diesen Koordinaten-Ursprung.
  Zeiger auf die ausgefüllten ChCoords-Strukturen befinden sich in der WTStdMsg-Struktur.

```
type r_ChCoords=record
        ChX     :array [1..6] of short;
        ChY     :array [1..6] of short;
     end;
```

Die PlayListEntry-Struktur fügt sich zu einer kompletten Liste zusammen, die die eigentliche Playliste darstellt. Soll eine Playliste vom Anfang bis zum Ende abgespielt werden, so sind dafür die Daten der PlayListEntry-Strukturen zu verwenden. Die Reihenfolge ihrer Verkettung entspricht dabei der Reihenfolge, in der die Playlistendaten abgespielt werden.

```
type PlayListEntry=^r_PlayListEntry;
type r_PlayListEntry=record
        BeginOffset,EndOffset   :long; - diese Offsets markieren die Sample-
                                         daten, die zu diesem Playlisten-
                                         eintrag gehören;
        Rate                    :long; - Amiga®-Playrate, in der die Sample-
                                         daten abgespielt werden müssen
        Delay,Time              :long; - werden derzeit noch nicht unter-
                                         stützt
        Loop                    :word; - wie oft die Sampledaten beim Ab-
                                         spielen wiederholt werden müssen
        VolumeL,VolumeR         :word; - Abspiellautstärken des linken und
                                         des rechten Kanals im Bereich von
                                         0 bis 64
        Name                    :string[30]; - ein frei wählbarer Name für
```

```
                                        diesen Playlisten-Eintrag
        BeforeEntry,NextEntry   :PlayListEntry - Zeiger auf die in der Liste
                                        vor und nach diesem Eintrag liegen-
                                        den PlayListEntry-Strukturen
   end;
```

Die folgende Struktur legt verschiedene Datenbereiche im Sample fest
- ähnlich den einzelnen Instrumenten in einem Tracker. Diese dienen einer
komfortableren Editierbarkeit einer Playliste, sind aber an sich eigentlich
nicht nötig, da eine Playliste bereits alle erforderlichen Daten in der
PlayListEntry-Liste speichert. Die PlayListPattern-Struktur bzw. -Liste
ist identisch mit der TimePattern-Liste. Der eigentlich verwirrende Name
beruht auf der Tatsache, das diese Liste ursprünglich nur für die Playliste
vorgesehen war.

```
type PlayListPattern=^r_PlayListPattern;
type r_PlayListPattern=record
        BeginOffset,EndOffset   :long; - diese Offsets markieren die Daten,
                                        die zu diesem Pattern gehören
        Name                    :string[30]; - ein frei wählbarer Name
        BeforePattern,NextPattern :PlayListPattern; - Zeiger auf die in der
                                        PlayListPattern-Liste vor und nach
                                        diesem Eintrag liegenden Strukturen
end;
```

Die WTStdMsg-Struktur wird beim Aufruf von Modulen übergeben und enthält
alle wichtigen Daten und Zeiger auf alle nötigen Informationen.

```
type r_WTStdMsg=^WTStdMsg;
type WTStdMsg=Record
        Private         :string[15]; - reserviert
        UNDOPossible    :boolean;    - UNDO-Speicher ist allociert und bereits
                                       mit Daten gefüllt
        Flags           :long;       - Kommandos an das Modul und Meldungen vom
                                       Modul
        Version         :long;       - aktuelle Version der WTStdMsg-Struktur,
                                       muß mit der Modulschnittstellenversion
                                       identisch sein
        WTScreen        :^Screen;    - Pointer auf den WaveTracer-Screen
        WTWindow        :^Window;    - Pointerauf das WaveTracer-Backdrop-
                                       window
        ActiveChannels :long;        - derzeit benutzte Kanäle
        ActiveMode      :long;       - aktueller Soundmode
        MemA16          :array [1..6] of long;
                                     - MemA16[1] und MemA16[2] sind die
                                       Startadressen der Speicherbereiche für
                                       die WaveTracer-internen 16Bit-Daten
                                       Länge: MemL24 / 2
        MemA24          :array [1..6] of long;
                                     - Startadressen aller Speicherbereiche
                                       für 24Bit-Daten
                                       Länge: MemL24
        MemAUNDO        :array [1..6] of long;
                                     - Startadressen aller Speicherbereiche
                                       für 24Bit-UNDO-Daten
```

```
                                Länge: MemL24
    * für diese drei Arrays gilt folgende Zuordnung: 1 - CH_LEFT, *
    * 2 - CH_RIGHT, 3 - CH_CENTER, 4 - CH_SLEFT, 5 - CH_SRIGHT,   *
    * 6 - CH_Sub                                                  *

MemL24          :long;       - Länge des für 24Bit-Daten reservier-
                               ten Speichers
PlayL24         :long        - Länge des mit 24Bit-Daten wirklich
                               belegten Speichers
    * Für die Umrechnung von 24Bit-Längenwerten, die ja eigent-   *
    * lich 32Bit-Werte darstellen, gelten folgende Divisoren:     *
    * von 24 Bit auf 16 Bit: Division durch 2                     *
    * von 24 Bit auf 8 Bit: Division durch 4                      *
SRate           :long;       - aktuelle Samplerate, in der das
                               Sample bei der betätigung der
                               "Space"-Taste abgespielt wird
Attack          :long;       - Startoffset des Attack-Bereiches
Decay           :long;       - Startoffset des Decay-Bereiches
Sustain         :long;       - Startoffset des Sustain-Bereiches
Release         :long;       - Startoffset des Release Bereiches
    * Die Daten, die sich im Adressbereich von Sustain bis        *
    * Release befinden, entsprechen dem zu loopenden Bereich und  *
    * damit dem vom WaveTracer unterstützten Sustain-Release-     *
    * Loop, dessen Ende nicht mit dem Sampleende identisch sein   *
    * muß. Alle anderen Offsets haben derzeit noch keine          *
    * Bedeutung. Die Offset-Werte gelten immer von der Start-     *
    * Adresse MemA24[x] des jeweiligen Kanals an.                 *
ActWaveOp       :^WaveOp;    - Zeiger auf die für die aktuelle
                               Funktion benutzte WaveOp-Struktur
                               des WTA-Scripts
OPos            :long;       - aktuelle Position im WTA-Script
OEnd            :long;       - erste Position im WTA-Script, die keine
                               gültigen Daten mehr enthält
AreaA           :long;       - Offset des Anfangs des momentan mar-
                               kierten Bereiches, gerechnet von
                               WTStdMsg.MemA24[x] an
AreaE           :long;       - Offset des Endes des momentan mar-
                               kierten Bereiches, gerechnet von
                               WTStdMsg.MemA24[x] an
ActFHandle      :BPTR;       - Filehandle einer bereits geöffneten
                               Datei bei Lade- und Speichermodulen
WTImg           :^WTImages;  - Zeiger auf die zu verwendenden Images
WTMsgPrc        :^r_MsgPrc;  - Zeiger auf die PRC_-Variablen
PosSurround     :^r_ChCoords; - Lautsprecherpositionen bei den
                               Soundmodi MD_MONO, MD_STEREO,
                               MD_3CH und MD_SURROUND
PosQuadro       :^r_ChCoords; - Lautsprecherpositionen beim Sound-
                               modus MD_QUADRO
PosAC3          :^r_ChCoords; - Lautsprecherpositionen dei den
                               Soundmodi MD_AC3_Sub und MD_AC3
UsedChannels    :long;       - gibt an, welche Kanäle derzeit Daten
                               enthalten, die von Null verschieden
                               sind.
MemAAlpha       :long;       - Startadressen aller Alphakanäle
                               Länge: MemL24 / 80
AlphaUNDO       :long;       - Startadressen der UNDO-Speicherbe-
                               reiche für die Alphakanäle
```

```
                                    Länge: MemL24 / 80
        DataValid      :long;       - hat dieses Longword den Wert $BADBAD,
                                    dann sind die folgenden Zeiger
                                    gültig - wenn nicht, kommuniziert das
                                    Modul mit einer WaveTracer-Version,
                                    die diese Zeiger noch nicht kennt
                                   (keine sehr elegante Methode um eine
                                    Abwärtskompatibilität zu erreichen,
                                    aber die Wahrscheinlichkeit, das das
                                    mal nicht funktioniert, liegt bei
                                    1 : 4.294.967.296)
        FirstPlayListPattern :PlayListPattern; - Zeiger auf den Kopf der
                                    PlayListPattern-Liste, wobei die
                                    ersten gültigen Pattern-Daten erst in
                                    der folgenden Struktur (FirstPlayList-
                                    Pattern^.NextPattern) zu finden sind;
                                    die PlayListPattern-Liste ist mit der
                                    TimePattern-Liste identisch
        FirstPlayListEntry :PlayListEntry; Zeiger auf den Kopf der eigent-
                                    lichen Playliste, die ersten gültigen
                                    Daten befinden sich hier ebenfalls
                                    erst in der Struktur FirstPlayList-
                                    Entry^.NextEntry
        SpecialData    :^SDheader; - Pointer auf den Anfang einer Liste mit
                                    speziellen Daten, die der WaveTracer
                                    in irgendeiner Form nutzt
        AnimPath       :str; - Zeiger auf einen String für den Pfad der
                                    animation, die im "Anim-Frames"-Fenster
                                    dargestellt wird. Ist der String leer,
                                    so ist keine Animation vorhanden
        Free1,Free2,Free3,Free4,Free5,Free6 :ptr - eine viel
                                    elegantere Methode, Abwärtskompa-
                                    tibilität zu erreichen
    end;

{$endif}
```

1.24 iff2eff

```
  Generierung von ".eff.data"-Dateien mit IFF2Eff
  -----------------------------------------------
```

Jedes Effektmodul benötigt eine Datei, die zum einen die Image-Daten für den grafischen Effektmodul-Requester und das WTA-Script und zum anderen die Informationen über die Art und die Behandlung des Effektmoduls beinhaltet.

Die Infos über das Modul befinden sich am Anfang der ".eff.data"-Datei in Form eines Longwords. Daran schließen sich die Imagedaten an.

Diese Datei kann mittels des mitgeliegferten Programmes "IFF2Eff" aus einer IFF-ILBM-Bilddatei erzeugt werden. Das Programm ist nur von CLI/Shell aus zu starten und verlangt dort auch die Eingabe des Bild-Dateinamens und der Flags. Das Bild muß 53 x 24 Punkte groß und 3 Bitplanes (=8 Farben) tief sein (das WaveTracer-interne Image-Handling erfolgt natürlich kompa-

tibel zu den bis zur Version Mark III verwendeten 4-Farben-Bildern). Werden Bilder anderer Größe benutzt, so gibt das keine Fehlermeldung, dafür aber Datensalat im Effektmodul-Requester des WaveTracer.

Das ist zugegebenermaßen alles andere als benutzerfreundlich, aber ich hatte 1. keine Zeit und keine Lust da etwas aufwendigeres zu programmieren und 2. sind wir ja keine normalen Menschen, sondern Programmierer. Sollte sich aber jemand beimachen und einen richtig schönen Konverter basteln, der auch für zukünftigen Erweiterungen auf 16 und mehr Farben geeignet ist, so wäre mir das mindestens eine kostenlose Vollversion wert!

1.25 sources

Die beiliegenden Beispielmodule

Um die Programmierung von Modulen zu verdeutlichen, habe ich für jeden Modultyp ein Beispielprogramm in Form von Quellcode mitgeliefert. Diese Dateien befinden sich im Verzeichnis "Sources".

Im Quellcode sind alle für die Programmierung relevanten Konstrukte erläutert.

Diese Beispiele, die Quellcodes originaler Module sind, sind nur als zusätzliche Erläuterung der Programmierung von Modulen anzusehen. Eine Erweiterung oder eine Verwendung dieser Quellcodes, um damit eigene, im Prinzip ähnliche Module zu erstellen ist nicht erlaubt.

1.26 pascalc

PASCAL - C - oderwasauchimmer

Im folgenden möchte ich eine kleine Übersetzungshilfe für die untereinander kompatiblen und in dieser Dokumentation verwendete Variablentypen von Pascal geben. Diese sind sich im Namen zu ihrenC-Pendants zwar recht ähnlich, weisen aber dennoch große Unterschiede auf.

Pascal	C	Beschreibung
byte	UBYTE	8 Bit vorzeichenlos
short	BYTE	8 Bit vorzeichenbehaftet
word	UWORD	16 Bit vorzeichenlos
integer	WORD	16 Bit vorzeichenbehaftet
long	LONG	32 Bit vorzeichenbehaftet
string	char[]	Zeichenkette
str	STRPTR	Pointer auf eine Zeichenkette

ptr	(evtl. APTR)	Pointer
boolean	BOOL	logische Variable kann TRUE oder FALSE sein
^		Pointer auf einen Variablentypen (z.B. ^long) oder auf eine Struktur (z.B. ^Screen)