SampleEditor v1.0

Plug-in-Tutorial

Inhaltsverzeichnis

1 Das Audioobjekt 3
2 Beschreibung der Plug-in-Schnittstelle 3
3 Implementierung eines Beispiel-Plug-in 5
4 Berücksichtigung einer Auswahl 8
5 Änderung des Audioformats 9
6 Erzeugen eines „leeren“ Audioobjekts 10
7 Erzeugen von Dialogen 10
7.1. Eigene Dialoge 10
7.2. Generischer Dialog (`GenericDialog` - *ImageJ*) 11
7.3. Generischer Dialog (`GenericDialog` - SampleEditor) 12
8 Ausgabe der Verarbeitung über ein Fenster 14
9 Kompilieren von Plug-ins über die Anwendung 14
10 Kompilieren über die Kommandozeile 16

1 Das Audioobjekt

Die vom *SampleEditor* verwalteten Audiodaten liegen im Datentyp `double` vor und werden beim öffnen von Audiodateien in einen Wertebereich von [-1 / +1] konvertiert. Ein Audioobjekt wird über eine Instanz der Klasse `AudioContent` repräsentiert. `AudioContent`-Objekte werden zur Verarbeitung an Plug-ins übergeben. Die wichtigsten Attribute des `AudioContent`-Objekts sind:

- **Sampledaten in Form eines mehrdimensionalen Arrays** (`double`)
- Das intern verwendete Audioformat der Audiodaten (`javax.sound.sampled.AudioFileFormat`)
- **Das Quellformat der Audiodaten** (`javax.sound.sampled.AudioFileFormat`)
- Ein `Selection`-Objekt, welchen eine **Benutzerauswahl** darstellt.

Die Dimensionen des mehrdimensionalen Sample-Arrays entsprechen den im Audiomaterial vorhandenen Audiokanälen:

Abb. 1: Abbildung der Audiokanäle im Sample-Array

2 Beschreibung der Plug-in-Schnittstelle

Die Plug-in-Schnittstelle `Processor` definiert Vorgaben, welche ein Plug-in erfüllen muss. Plug-ins müssen diese Schnittstelle realisieren. Nachfolgend werden die zu implementierenden Methoden erläutert.

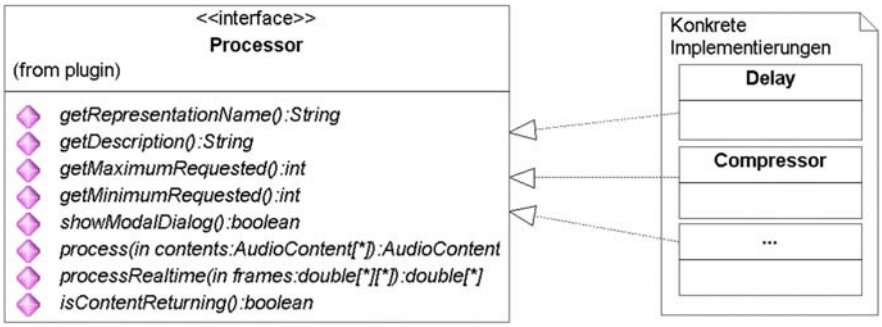

Abb. 2: UML-Diagramm des Plug-in Interface

- **`getRepresentationName():String`**
  Der Titel des Plug-in. Dieser wird im Menü der Anwendung angezeigt.

- **`getDescription():String`**
  Der Autor eines Plug-in hat hier die Möglichkeit die Funktionalität des Plug-in in kurzer Form zu beschreiben. Die Beschreibung wird in der Statusleiste der Anwendung angezeigt.

- **`getMaximumRequested():int`**
  Maximale Anzahl der zur Verarbeitung benötigten `AudioContent`-Objekte. Hier können Werte im Bereich von `0` bis `n`, oder `-1` für eine unbestimmte Anzahl angegeben werden. Abhängig vom Rückgabewert aus der Methode `getMinimumRequested()` wird ein Dialog zur Auswahl von Audioobjekten für den Benutzer generiert.

- **`getMinimumRequested(): int`**
  Mindestanzahl der zur Verarbeitung benötigten `AudioContent`-Objekte. Hier können Werte im Bereich von 0 bis n angegeben werden. Die Methode dient zur Kommunikation zwischen Anwendungen und Plug-in. Abhängig vom Rückgabewert aus der Methode `getMaximumRequested()` wird ein Dialog zur Auswahl von Audioobjekten für den Benutzer generiert.

  Beispiele:
  Genau 1 Audioobjekt: Maximum: 1, Minimum: 1
  Genau 3 Audioobjekte: Maximum: 3, Minimum: 3
  Mindestens 2, unendlich viele weitere Audioobjekte: Maximum: -1, Minimum: 2
  Mindestens 1, höchstens 3 Audioobjekte: Maximum: 3, Minimum: 1

- **`isContentReturning(): boolean`**
  Über einen booleschen Wert muss hier angegeben werden ob das Plug-in nach der Verarbeitung ein `AudioContent`-Objekt zurückliefert (`true`) oder nicht (`false`). Die Methode dient zur Kommunikation zwischen Anwendungen und Plug-in. Beim Rückgabewert `false` wird von der Anwendung angenommen, dass das Plug-in eine mögliche Ausgabe selbst übernimmt. Das bedeutet, dass der Rückgabewert der Verarbeitungsmethode `process()` von der Anwendung nicht berücksichtigt wird.

- **`showModalDialog(): boolean`**
  Diese Methode kann vom Autor eines Plug-in zur Implementierung modaler Benutzerdialoge verwendet werden. Die Methode wird von der Anwendung in jedem Fall vor der Verarbeitungsmethode `process()` aufgerufen. Anhand des Rückgabewerts (`boolean`) wird entschieden, ob der Verarbeitungsvorgang gestartet oder abgebrochen wird.

- **`process(AudioContent[] contents): AudioContent`**
  Hier erfolgt die Implementierung der Funktionalität des Plug-in. Beim Aufruf der Methode durch die Anwendung wird ein Array von `AudioContent`-Objekten übergeben, welches 0 bis n Elemente enthalten kann. Der Rückgabewert kann ein `AudioContent`-Objekt oder `null` sein. Das `AudioContent`-Objekt mit dem Array-Index 0 hat eine besondere Bedeutung. Es repräsentiert das im Editor aktuell ausgewählte Audioobjekt.

- **`processRealtime(double[][] frames): double[]`**
  Die Methode kann in einer `Processor`-Implementierung verwendet werden, um echtzeitfähigen Programmcode im Nicht-Echtzeit-Betrieb zu verwenden.

3 Implementierung eines Beispiel-Plug-in

Im Folgenden wird die Implementierung eines einfachen Plug-in erläutert. Das Plug-in erhöht lediglich die Amplitude der Audiodaten, indem alle vorhandenen Samplewerte mit dem Faktor 2.0 multipliziert werden. Diese Manipulation verändert den Pegel des Signals und äußert sich bei der Audiowiedergabe in einer Lautstärkenveränderung. Das Plug-in ist darauf ausgelegt die Audiodaten von genau einem Audioobjekt zu manipulieren und liefert dieses nach der Verarbeitung an die Anwendung zurück. Im Folgenden wird die Implementierung der Methoden der Plug-in-Schnittstelle `Processor` erläutert.

```
public class ExampleProcessor implements Processor
{
```

Der Name des Plug-in.

```
public String getRepresentationName()
{
      return "Example Volume Processor";
}
```

Eine kurze Beschreibung der Funktionalität.

```
public String getDescription()
{
      return "Increases the signals volume (factor 2.0)";
}
```

Das Plug-in soll genau ein Audioobjekt verarbeiten. Um diese Anzahl von der Applikation zu fordern, müssen die Methoden `getMaximumRequested()` und `getMinimumRequested()` beide den Wert 1 zurückgeben. Wird genau ein Audioobjekt von der Anwendung gefordert, liefert diese das vom Benutzer aktuell ausgewählte Audioobjekt (Aktives Fenster).

```
public int getMaximumRequested()
{
      return 1;
}
```

```
public int getMinimumRequested()
{
      return 1;
}
```

Das Plug-in verarbeitet die eingehenden Audiodaten und liefert das manipulierte Audioobjekt an die Anwendung zurück. Die Methode `isContentReturning()` muss deshalb den Wert `true` zurückgeben. Angenommen ein Plug-in manipuliert keine Audiodaten, sondern verarbeitet die Samplewerte in einem Histogramm, dann wird kein Audioobjekt an die Anwendung zurückgegeben. In diesem Fall muss die Methode den Wert `false` liefern.

```
public boolean isContentReturning()
{
	return true;
}
```

In der Methode `showModalDialog()` können Dialoge Implementiert werden, über die ein Benutzer verschiedene Einstellungen vornehmen kann. Im Beispiel-Plug-in soll kein Dialog angezeigt werden. Deshalb muss hier der Wert `true` zurückgeliefert werden. Würde `false` zurückgegeben werden bricht die Anwendung den Vorgang ab.

```
public boolean showModalDialog()
{
	return true;
}
```

Die Methode `processRealtime()` muss zwar realisiert werden, kann aber den Wert null zurückliefern. Die Methode hat in diesem Fall keine Bedeutung.

```
public double[] processRealtime(double[][] frames)
{
	return null;
}
```

In der Methode `process()` findet die Verarbeitung statt. Beim Aufruf der Methode übergibt die Anwendung ein `AudioContent`-Array welches grundsätzlich 0 bis n Objekte enthalten kann. Über die Methoden `getMaximumRequested()` und `getMinimumRequested()` wurde zuvor genau ein Audioobjekt von der Anwednung gefordert. Deshalb kann davon ausgegangen werden, dass das übergebene Array genau ein Element mit dem Array-Index 0 enthält.

```
public AudioContent process(AudioContent[] content)
{
//Referenz auf das AudioContent-Objekt
AudioContent audioContent = content[0];
//Referenz auf das Sample-Array
double[][] samples = audioContent.getSamples();

//Anzal der Kanäle und Samples
int channels = samples.length;
int totalLength = samples[0].length;

//Gain-Faktor
double gain = 2.0;

//Schleife über die Sampleanzahl
for (int sampleIndex = 0; sampleIndex < totalLength; sampleIndex ++)
{
      //Schleife über die Kanäle
      for (int channelIndex = 0; channelIndex < channels; channelIndex++)
      {
            //Aktuelles Sample auslesen
            double sample = samples[channelIndex][sampleIndex];
            //Verarbeitung: Samplewert mit dem Gain-Faktor multiplizieren
            sample *= gain;
            //Clipping: Bei der Multiplikation kann es vorkommen, das die
            Samplewerte außerhalb des Wertebereichs von [-1 / +1] liegen.
            Ist das der Fall wird ein maximaler Wert -1 oder +1 gesetzt.
            sample = sample > 1 ? 1 : sample;
            sample = sample < -1 ? -1 : sample;
            //Zurückschreiben des Samplewerts
            samples[channelIndex][sampleIndex] = sample;
      }
}
//Rückgabe des manipulierten AudioContent-Objekts
return audioContent;
}
}
```

4 Berücksichtigung einer Auswahl

Um die Verarbeitung des Plug-ins auf eine vom Benutzer getroffene Auswahl beschränken zu können, steht in einer `AudioContent`-Instanz ein `Selection`-Object (Auswahl) zur Verfügung. Eine Auswahl besitzt die Attribute Startframe und Endframe und ist „leer“, wenn beide Attribute den gleichen Wert enthalten. Abgefragt werden kann dies über die Methode `isEmpty()`, welche `true` zurückgibt, wenn Start- und Endframe gleich sind.

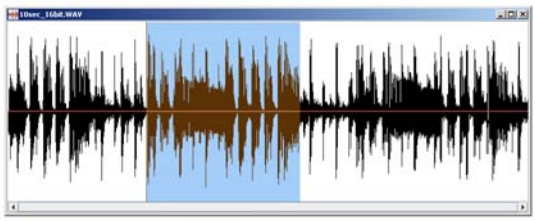

Abb. 3: Ausgewählter Teilbereich im Graphen

Beispiel:

```
public AudioContent process(AudioContent[] content)
{
	//Referenz auf das AudioContent-Obekt
	AudioContent audioContent = content[0];

	//Referenz auf die Auswahl
	Selection selection = audioContent.getSelection();

	//Indices für den Schleifendurchlauf
	int sampleIndex = 0; //Initialisierung: Erster Sample-Index
	int endIndex = samples[0].length; //Initialisierung: Gesamtlänge

	//Ändern der Indices, wenn eine Auswahl vorhanden ist
	if(!selection.isEmpty())
	{
		sampleIndex = selection.getStartFrame();
		endIndex = selection.getEndFrame();
	}

	for ( ; sampleIndex < endIndex; sampleIndex++)
	{
		... //process
	}
	return audioContent;
}
```

5 Änderung des Audioformats

Wird durch ein Plug-in das Audioformat eines Audioobjekts verändert, dann ist es erforderlich das neue Format im `AudioContent`-Objekt zu setzten. Bei einer Manipulation der Amplitudenwerte dies nicht notwendig. Werden hingegen die Anzahl der Kanäle, die Länge der Audiodaten (Anzahl der Samples) oder die Samplerate verändert, ist dies zwingend erforderlich.

Zulässige Änderungen des Audioformats:

- **Veränderung der Kanalanzahl**
- **Veränderung der Samplerate**
- **Veränderung der Sampleanzahl (Länge des Audioobjekts)**

Unzulässige Änderungen des Audioformats (Innerhalb der Anwendung festgelegte Werte!):

- **Veränderung der Bit-Tiefe (Samplesize: 16 Bit)**
- **Veränderung der Kodierung (PCM_SIGNED)**
- **Veränderung der Byte-Order (Endian: LITTLE_ENDIAN)**

Das intern verwendete Audioformat wird in einer `AudioContent`-Instanz über ein Objekt der Klasse `javax.sound.sampled.AudioFileFormat` beschrieben. Die Erzeugung des Objekts ist etwas umständlich, weshalb in der Klasse `de.fhtw.sampleeditor.util.AudioUtil` einige statische Methoden zur Vereinfachung bereitgestellt werden. Die Methoden verlangen als Übergabeparameter das ursprüngliche `AudioFileFormat` und die entsprechenden Veränderungen. Der Rückgabewert ist ein neues `AudioFileFormat`-Objekt, welches die angegebenen Änderungen enthält.

- `getAudioFileFormat_Ch(AudioFileFormat oldFileFormat, int newChannels)`
- `getAudioFileFormat(AudioFileFormat oldFileFormat, int newFrameLength)`
- `getAudioFileFormat(AudioFileFormat oldFileFormat, int newSampleRate, int newFrameLength)`
- `getAudioFileFormat(AudioFileFormat oldFileFormat, int newSampleRate, int newFrameLength, int newChannels)`

Das intern verwendete `AudioFileFormat` kann im `AudioContent`-Objekt über folgende Methoden angesprochen und neu gesetzt werden:

```
AudioFileFormat iaff = audioContent.getInternalAudioFileFormat()

audioContent.setInternalAudioFileFormat(AudioFileFormat newFormat);
```

6 Erzeugen eines „leeren“ Audioobjekts

Plug-ins könnten ein neues Audioobjekt benötigen. (z. B. ein Tongenerator). Ein „leeres“ Audioobjekt kann über die Klasse `de.fhtw.sampleeditor.util.AudioUtil` erzeugt werden. Hierfür stehen zwei statische Methoden zur Verfügung:

- **`getEmptyAudioContent`**`(`**`int`** `seconds)`
  Liefert ein `AudioContent`-Objekt mit der angegebenen Länge in Sekunden im Standartformat: *44.100 Hz, 16 Bit, PCM_SIGNED, LITTLE_ENDIAN, Stereo*

- **`getEmptyAudioContent`**`(`**`int`** `seconds,` **`int`** `sampleRate,` **`int`** `channels)`
  Liefert ein `AudioContent`-Objekt mit der angegebenen Länge in Sekunden, der Samplerate und der Anzahl der Kanäle. In der aktuellen Version der Anwendung werden allerdings nur Audioobjekte mit einem oder zwei Kanälen unterstützt. Die Methode liefert in jedem Fall Mono- oder Stereodaten (`channels` = 0 → Mono, `channels` > 2 → Stereo).

Die Bit-Tiefe, die Kodierung und die Byte-Order können nicht angegeben werden, da diese Werte innerhalb der Anwendung festgelegt sind.

7 Erzeugen von Dialogen

Um dem Benutzer die Möglichkeit zu geben verschiedene Parameter des Plug-in einstellen zu können, werden Benutzerdialoge benötigt. Benutzerdialoge sind in der Methode `showModalDialog()` zu implementieren. Es ist zu beachten, dass ein Benutzer generell die Möglichkeit haben sollte den Verarbeitungsvorgang abzubrechen. Deshalb muss die Methode einen booleschen Wert zurückgeben (`true` oder `false`). Über den booleschen Wert entscheidet die Anwendung ob die Verarbeitung abgebrochen oder durchgeführt wird. Innerhalb der Methode können eigene Dialoge implementiert oder vorhandene generische Dialoge verwendet werden.

7.1. Eigene Dialoge

Bei der Implementierung eigener Dialoge muss berücksichtigt werden, dass diese in jedem Fall ***modal*** sind und somit den Programmablauf blockieren. Werden keine modalen Dialoge verwendet, erfolgt zwar die Anzeige des Dialogs, aber die Verarbeitung wird unmittelbar durchgeführt. Das bedeutet, dass der Benutzer keine Einstellungen über den Dialog vornehmen kann, da die Verarbeitung bereits ausgeführt wurde.

7.2. Generischer Dialog (`GenericDialog` - *ImageJ*)

Um die Erstellung von Dialogen zu vereinfachen, wurde die Klasse `GenericDialog` aus dem Bildbearbeitungsprogramm *ImageJ* übernommen. Über die Klasse ist es möglich, mit relativ wenig Aufwand, einen anpassbaren, modalen Dialog zu erzeugen.

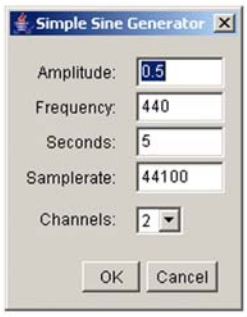

Abb. 4: Generischer Dialog - ImagerJ

Für den Benutzer einstellbare Plug-in-Parameter können über Textfelder, Schieberegler, Auswahlboxen und andere grafische Bedienerelemente dargestellt werden. Die verschiedenen Möglichkeiten sind der API-Dokumentation, unter: *http://rsb.info.nih.gov/ij/developer/api/ij/gui/GenericDialog.html* zu entnehmen. Innerhalb der Anwendung kann der generische Dialog über die Klasse `GenericDialog` verwendet werden:

```
de.fhtw.sampleeditor.gui.dialog.generic.ij.GenericDialog
```

Beispiel:

```
…
//Beispielwerte
double amplitude = 0.5;
double frequency = 440;
int seconds = 5;
int sampleRate = 44100;
int channels = 2;

…
public boolean showModalDialog()
{
      //Generischen Dialog erzeugen
      GenericDialog gd = new GenericDialog("Simple Sine Generator");
      //Dem Dialog Felder hinzufügen
      gd.addNumericField("Amplitude: ", amplitude,1);
      gd.addNumericField("Frequency: ", frequency, 0);
      gd.addNumericField("Seconds:", seconds, 0);
      gd.addNumericField("Samplerate: ", sampleRate, 0);
      String[] choices = {"1","2"};
      gd.addChoice("Channels: ", choices, "2");
      //Dialog anzeigen
      gd.showDialog();
      //Abbruch durch den Benutzer
```

```
        if (gd.wasCanceled()) return false;
        //Werte setzen
        amplitude = gd.getNextNumber();
        frequency = gd.getNextNumber();
        seconds = (int)gd.getNextNumber();
        sampleRate = (int)gd.getNextNumber();
        channels = (int)gd.getNextNumber();
        channels = Integer.parseInt(gd.getNextChoice());
        //Kein Abbruch durch den Benutzer
        return true;
}
```

7.3. Generischer Dialog (`GenericDialog` - SampleEditor)

Ein zweite Möglichkeit zur Erzeugung eines generischen Dialogs besteht über die Klasse `de.fhtw.sampleeditor.gui.dialog.generic.GenericDialog`. Allerdings können in der aktuellen Version nur Schieberegler generiert werden.

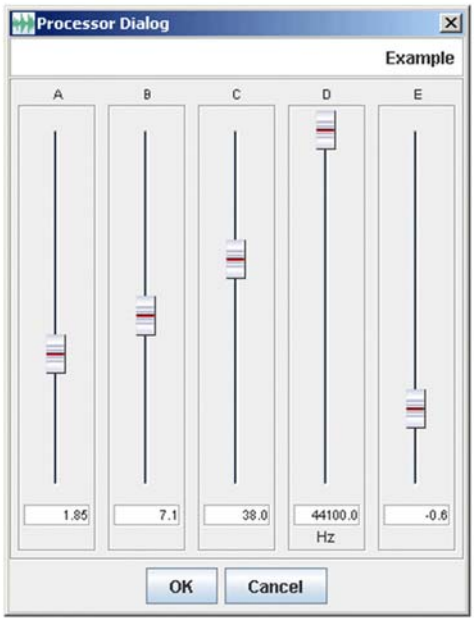

Abb. 5: Generischer Dialog - SampleEditor

In zukünftigen Versionen soll der generische Dialog erweitert werden, so dass verschiedene grafische Bedienerelemente dargestellt werden können. Der Konstruktor der Klasse `GenericDialog` verlangt ein `Control`-Array als Parameter. Ein `Control`-Objekt stellt ein Wertemodell dar, wobei bisher nur die Klasse `FloatRangeControl` für Gleitkommawerte zur Verfügung steht:

```
FloatRangeControl(String name, String unit, float minimum, float maximum,
float precision, float initValue)
```

Parameter	Beschreibung
name:	Beschriftung (Gain, Samplerate, Delaytime, etc.)
unit:	Einheit (Hz, %, etc.)
minimum:	Minimaler Wert
maximum:	Maximaler Wert
precision:	Genauigkeit
	0.01 = Zwei Stellen nach dem Komma
	0.1 = Eine Stelle nach dem Komma
	1 = Ganze Zahl
initValue:	Wert der Initialisierung

Tabelle 1: FloatRangeControl – Beschreibung der Parameter

Beispiel:

```
...

//Beispielwerte
double a = 2.47;
double b = 2;
double c = 50;
int d = 44100;
double e = -0.5;
//Erzeugen der verschiedenen Control-Objekte
FloatRangeControl aControl =
new FloatRangeControl("A","",0.0f,5.0f,0.01f,(float)a);
FloatRangeControl bControl =
new FloatRangeControl("B","",0.0f,15.0f,0.1f,(float)b);
FloatRangeControl cControl =
new FloatRangeControl("C","",0.0f,60.0f,1f,(float)c);
FloatRangeControl dControl =
new FloatRangeControl("D","Hz",22050.0f,44100.0f,1f,(float)d);
FloatRangeControl eControl =
new FloatRangeControl("E","",-1.0f,1.0f,0.1f,(float)e);

...

public boolean showModalDialog()
{
      //Erzeugen des Control-Arrays
      Control[] controls =
            {aControl, bControl, cControl, dControl, eControl};
      //Erzeugen des generischen Dialogs
      GenericSliderDialog dialog =
            new GenericSliderDialog("Example", controls);
      //Abbruch durch den Benutzer
      if(!dialog.getUserDecision()) return false;
      //Werte setzen
      a = aControl.getValue();
      b = bControl.getValue();
      c = cControl.getValue();
      d = (int)dControl.getValue();
      e = eControl.getValue();
      //Kein Abbruch durch den Benutzer
      return true;
}
```

8 Ausgabe der Verarbeitung über ein Fenster

Plug-ins, welche Audiodaten nicht manipulieren, sondern eventuell nur analysieren und über ein grafischen Fenster darstellen wollen, müssen die Ausgabe selbst übernehmen. Dabei ist zu beachten, dass die Plug-in-Klasse nicht von `java.awt.Frame`, `javax.swing.JFrame` oder entsprechenden Dialogen erben darf:

```
public class Example extends JFrame implements Processor
{
	public Example()
	{
		...
		setVisible(true);
	}
	...
}
```

Bei der Initialisierung der Anwendung werden Instanzen der Plug-ins erzeugt, was in diesem Fall dazu führt, dass ein Plug-in-Fenster bereits angezeigt wird. Die Ausgabe in einem grafischen Fenster sollte wie folgt o. ä. umgesetzt werden:

```
public class Example implements Processor
{
...
	public AudioContent process(AudioContent[] content)
	{
		//Verarbeitung...
		...
		//GUI
		JFrame frame = new JFrame();
		frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		frame.getContentPane().setLayout(...);
		...
		frame.getContentPane().add(...);
		frame.setSize(...);
		frame.setVisible(true);
		return null;
	}
}
```

9 Kompilieren von Plug-ins über die Anwendung

Plug-ins, welche in Form von Java-Quelltext vorliegen, können direkt über die Anwendung kompiliert und ausgeführt werden. Hierzu muss der Quelltext im Plug-in-Verzeichnis (*SampleEditor/plugin*), oder in dessen Unterverzeichnissen abgelegt werden (siehe Abb. 1: *ExampleProcessor.java*). Dateien außerhalb des Verzeichnis werden nicht kompiliert und ausgeführt.

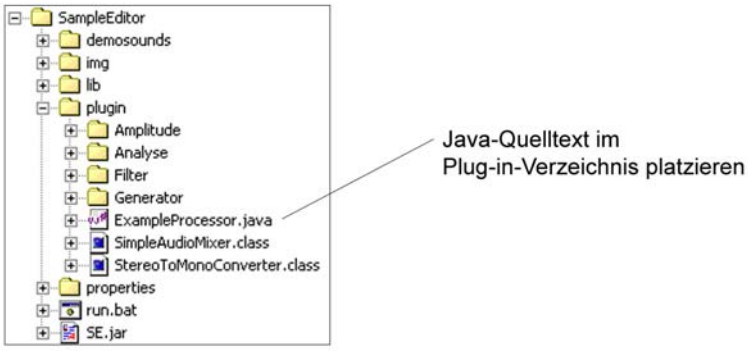

Abb. 6: Platzieren von Java-Quelltext im Plug-in-Verzeichnis

Über den Menüpunkt ‚*Plugins/Compile & run...*' öffnet sich zunächst ein Auswahldialog. Hier wird der auszuführende Plug-in-Quelltext ausgewählt.

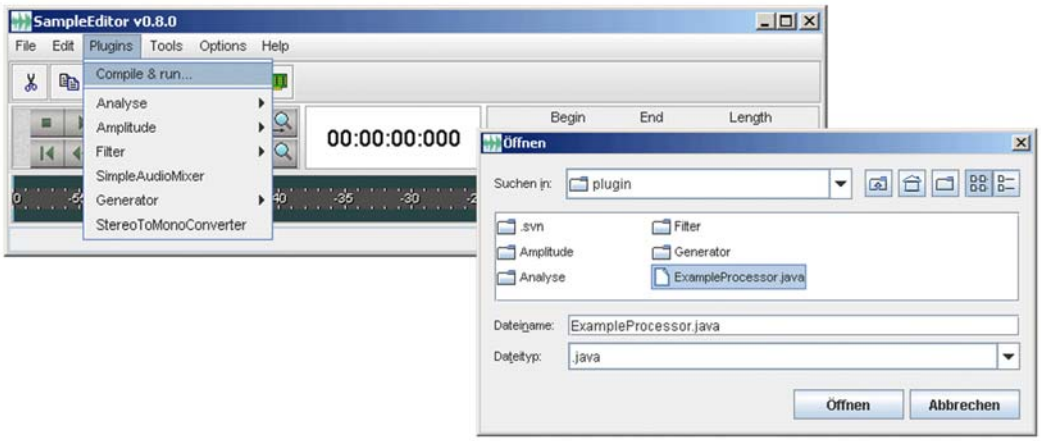

Abb. 7: Kompilieren von Plug-ins über die Anwendung

Ist der Quelltext fehlerfrei, wird das Plug-in kompiliert, geladen und umgehend ausgeführt. Im Plug-in-Menü erscheint das Plug-in erst nach Neustart der Anwendung. Sind Fehler vorhanden, erfolgt die Ausgabe der Compiler-Fehlermeldung in einem neuen Fenster (siehe Abb. 8).

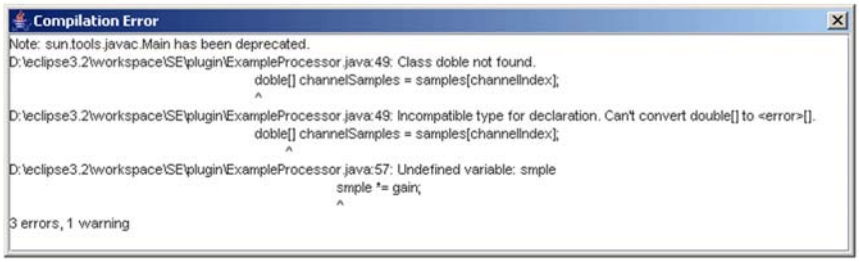

Abb. 8: Fehlermeldung des *Javac*-Kompilers

10 Kompilieren über die Kommandozeile

Die Kompilierung von Plug-ins kann auch außerhalb der Anwendung über die Kommandozeile erfolgen. Hierfür muss dem *Javac*-Compiler über die Option *-classpath* der Pfad zum *Jar*-Archiv der Anwendung (*SE.jar*) mitgeteilt werden. Im Beispiel auf der Abb. 9 wurden der Quelltext und das *Jar*-Archiv im Verzeichnis *pluginTest* abgelegt. Der Aufruf des *Javac*-Compilers kann dann über den Befehl: `javac ExampleProcessor.java -classpath SE.jar` erfolgen. Die vom Compiler erzeugte `Class`-Datei kann im Plug-in-Verzeichnis der Anwendung platziert werden und ist beim Neustart der Anwendung als Plug-in verfügbar. Besteht ein Plug-in aus mehreren Klassen, müssen demenstprechend alle kompilierten `Class`-Dateien in das Plug-in-Verzeichnis kopiert werden. Plug-ins in Form von gepackten *Jar*-Archiven werden in dieser Version nicht unterstützt.

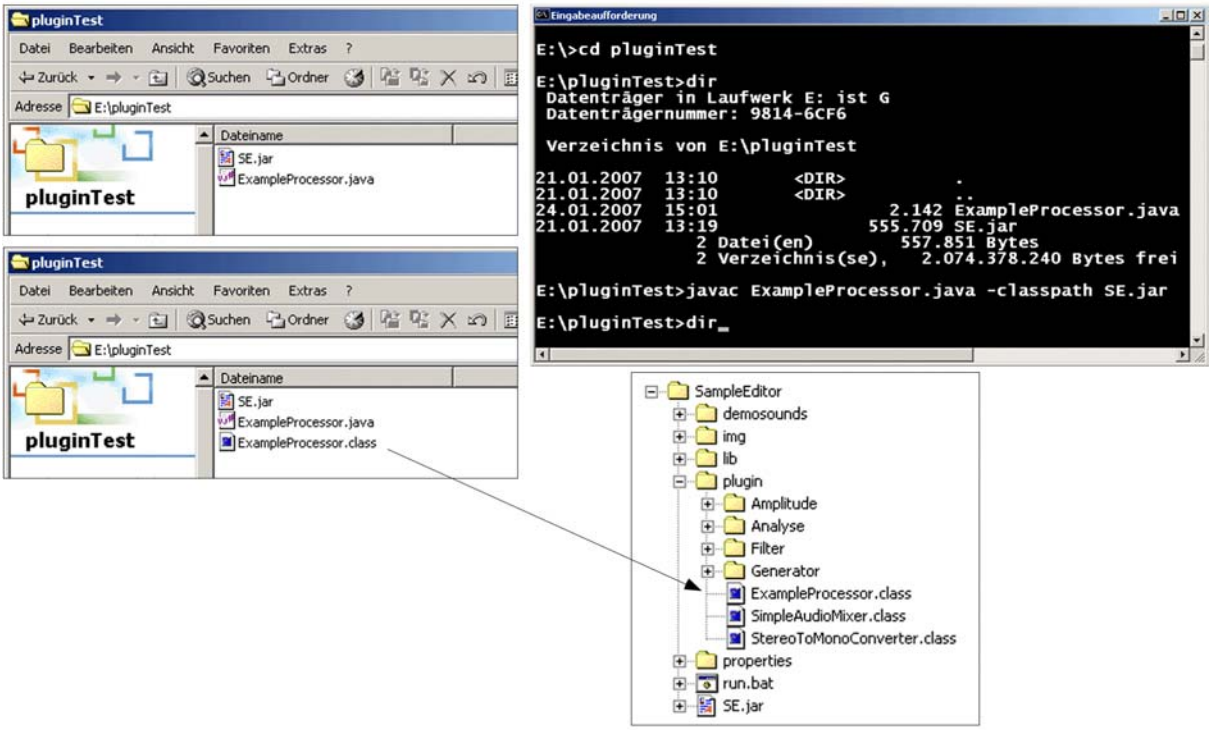

Abb. 9: Kompilierung über die Kommandozeile