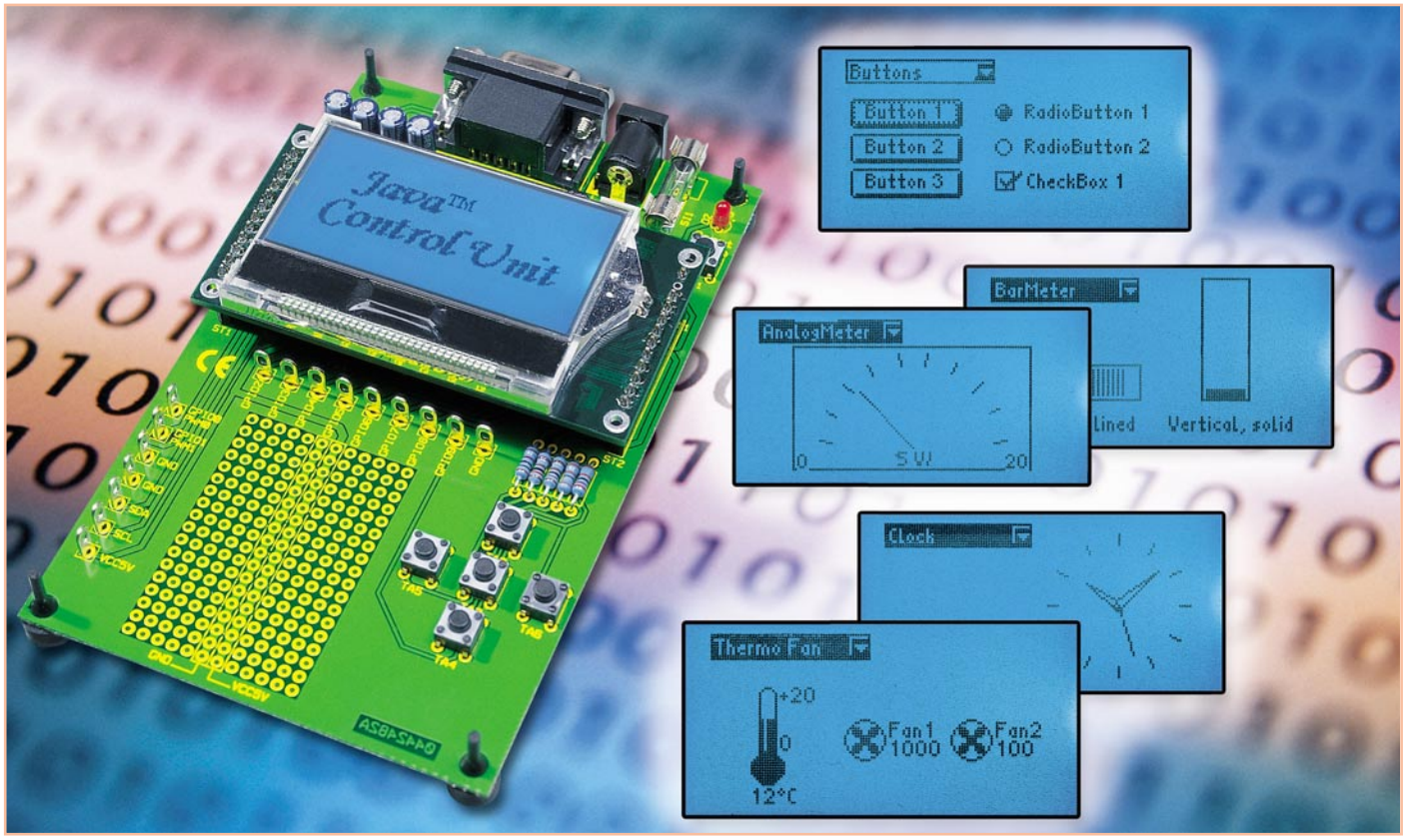
Steuern und visualisieren –



Java™-Control-Unit JCU 10 mit JControl-Technologie Teil 3

Nachdem wir in der letzten Ausgabe die Hardware der Java™-Control-Unit vorgestellt haben, werden wir uns in diesem Teil mit der Programmierung beschäftigen. Hier wollen wir zunächst beschreiben, wie man die fertigen Beispielprojekte in den Speicher der Java™-Control-Unit lädt. Danach zeigen wir anhand eines einfachen Beispielprogramms, wie man aus selbst erstellten Java-Quellen mit der JControl/IDE ein neues Projekt realisiert.

Installation der JControl/IDE

Die Installation der JControl/IDE erfolgt in wenigen einfachen Schritten:

- die mit der JCU10 mitgelieferte CD in das Laufwerk einlegen,
- wenn die Installation daraufhin nicht automatisch beginnt, ist die EXE-Datei auf der CD zu starten,
- im ersten Dialogfenster der Installation wird die gewünschte Sprache ausgewählt,
- dann folgt man den weiteren Anweisungen in den Dialogfenstern und akzeptiert die Lizenzbestimmungen,
- anschließend erfolgt die Auswahl des Zielordners für die Installation und die des Standard-Installationssatzes,
- abschließend wählt man die Verknüpfungsposition aus – wie vorgegeben in einer neuen Programmgruppe,
- überprüft im dann folgenden Dialogfenster die vorgenommenen Einstellungen und startet mit dem Button „Installieren“ die Installation.

Nach der erfolgreichen Installation mit der vorgegebenen Verknüpfungsposition finden sich in der Programmgruppe JControl folgende Einträge:

- **API-Dokumentation:** Beschreibung der verschiedenen Packages und der darin enthaltenen Klassen im HTML-Format
- **Ordner „Examples“:** eine Anzahl von fertigen Projekten mit Programmbeispielen
- **PropertyEdit:** Hilfsprogramm, mit dem Einstellungen der Java™-Control-Unit ausgelesen und geändert werden können
- **Uninstall JControl:** Deinstallation der JControl-Software
- **IDE:** die integrierte Entwicklungsumgebung

JControl/IDE

Fertige Projekte in die JCU 10 laden

Wir beginnen zum Kennenlernen der Entwicklungsumgebung zunächst damit, ein fertiges Programmbeispiel zu öffnen und in den Speicher der Java™-Control-Unit zu übertragen. Bei der Installation der

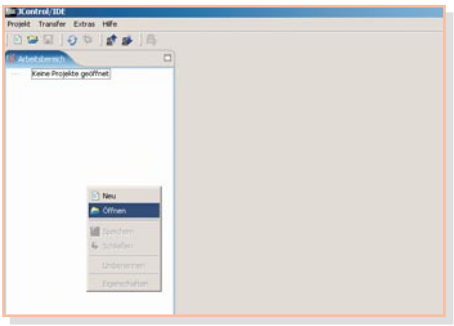


Bild 12: Durch einen Klick mit der rechten Maustaste in den Arbeitsbereich lassen sich Projekte öffnen oder neu erstellen.

JControl/IDE wurde im Installationsverzeichnis der Ordner *examples* erstellt, der eine ganze Reihe von Beispielprojekten enthält. Ein solches Projekt wollen wir nun in den Speicher der Java™-Control-Unit laden.

Wird die JControl/IDE zum ersten Mal aufgerufen, startet das Programm mit einer leeren Arbeitsfläche, dem Arbeitsbereich.

Das Öffnen eines Projektes erfolgt entweder über das Menü *„Projekt“* und den Menüpunkt *„Öffnen“* oder durch einen Klick mit der rechten Maustaste in den Arbeitsbereich, wie in Abbildung 12 dargestellt. In dem dann erscheinenden Popup-Menü wählt man den Punkt *„Öffnen“* aus.

Im folgenden Dialogfenster kann dann die Auswahl einer Projektdatei erfolgen. Wir haben uns an dieser Stelle für das Projekt *„VoleDemo.jcp“* entschieden, in dem etliche Komponenten des so genannten Vole-Frameworks dargestellt werden.

Das Projekt ist imVerzeichnis *„JControl\examples\VoleDemo“* zu finden.

Nach dem Öffnen des Projektes wird im Arbeitsbereich der Name des geöffneten Projektes angezeigt, und neben dem Arbeitsbereich sind im Quellen-Navigator die zu diesem Projekt gehörenden Quelldateien aufgelistet. In diesem Fall sind dies zwei Schriftarten, drei Bilder und der Java-Quelltext, wie in Abbildung 13 dargestellt.

Durch einen Doppelklick auf eine Datei im Quellen-Navigator öffnet sich unter dem Quellen-Navigator ein weiteres Fenster mit dem entsprechenden Editor für den jeweiligen Quellentyp. Durch einen Doppelklick auf den Java-Quelltext öffnet sich der Java Compiler, siehe Abbildung 14.

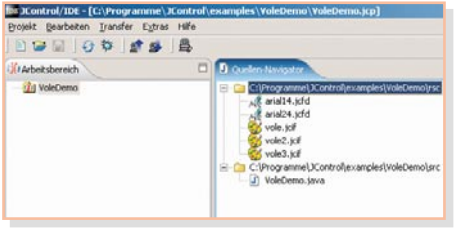


Bild 13: Das Projekt „VoleDemo“ mit den zugehörigen Quelldateien im Quellen-Navigator

Hier können Änderungen im Quelltext vorgenommen werden.

Beim Speichern der geänderten Datei durch einen Klick auf das Disketten-Symbol in der Symbolleiste (oder alternativ durch das Auswählen des Eintrags *„Speichern“* aus dem Menü *„Datei“*) wird der Quelltext neu kompiliert. Kommt es beim Kompilieren zu Problemen, erfolgt unterhalb des Quelltextes die Anzeige einer Liste mit Warnungen und Fehlermeldungen, die eine Beschreibung des Fehlers und die Zeilennummer enthalten. Durch einen Doppelklick auf eine Fehlermeldung gelangt man automatisch an die entsprechende Stelle im Quelltext.

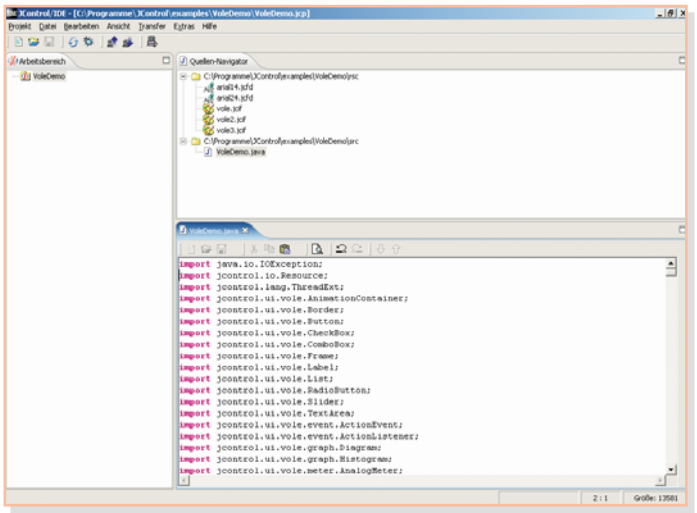


Bild 14: JControl/IDE mit geöffneter Java-Quelldatei

Möchte man statt des Java-Quelltextes eines der Bilder editieren, so öffnet sich nach einem Doppelklick auf die entsprechende Bild-Datei im Quellen-Navigator ein Zeichenprogramm, mit dem Änderungen am Bild vorgenommen werden können, siehe Abbildung 15.

Beim Auswählen von Schriftarten oder Melodien öffnet sich ebenfalls jeweils ein entsprechendes Programm, mit dem man die Quelldatei bearbeiten kann.

Um das Projekt in den Speicher der Java™-Control-Unit laden zu können, ist zunächst die Java™-Control-Unit in das Evaluation-Board einzusetzen und mit einem 9-poligen Sub-D-Verlängerungskabel (1 x Stecker, 1 x Buchse) mit der seriellen Schnittstelle des PCs zu verbinden.

In der JControl/IDE wählt man im Menü *„Extras“* unter *„Verbindungseinstellungen“* den entsprechenden COM-Port aus.

Als Nächstes muss nun die Java™-Control-Unit in den Download-Mode gebracht werden. Dazu drückt man zunächst die „Select“- und dann zusätzlich kurz die „Reset“-Taste des Evaluation-Boards. Die Java™-Control-Unit zeigt in diesem manuell gestarteten Download-Mode, wie schon in Teil 2 beschrieben, die Versionsnummer und die Übertragungsparameter an.

In der JControl/IDE startet man jetzt entweder durch Auswählen des Eintrags *„Projekt hochladen“* aus dem Menü *„Transfer“* (oder durch das Drücken des „Upload“-Buttons in der Symbolleiste) die Übertragung des Projektes an das angeschlossene Gerät.

Nachdem das Projekt vollständig übertragen ist, wird durch Drücken der „Reset“-Taste des Evaluation-Boards die Anwendung gestartet. Nachdem zunächst kurz das „animierte“ Bild der Wühlmaus (engl. vole) angezeigt wird, erscheinen vier Auswahlboxen, zwischen denen man mit den Richtungstasten wechseln kann. Um eine der Auswahlboxen zu aktivieren, drückt man die „Select“-Taste auf dem Evaluation-Board. Die Auswahlboxen zeigen dann ihre Einträge, zwischen denen man mit der „Up“- und „Down“-Taste wechseln kann. Wählt man einen Eintrag mit der „Select“-Taste aus, so wird das zugehörige Demo in der unteren Hälfte des Displays angezeigt.

Eigene Projekte und Quelltexte erstellen

Nachdem wir so die „handwerklichen“ Grundlagen von IDE und Datentransfer kennen gelernt haben, wollen wir als Nächstes ein neues Projekt erstellen und einen eigenen Quelltext darin anlegen.

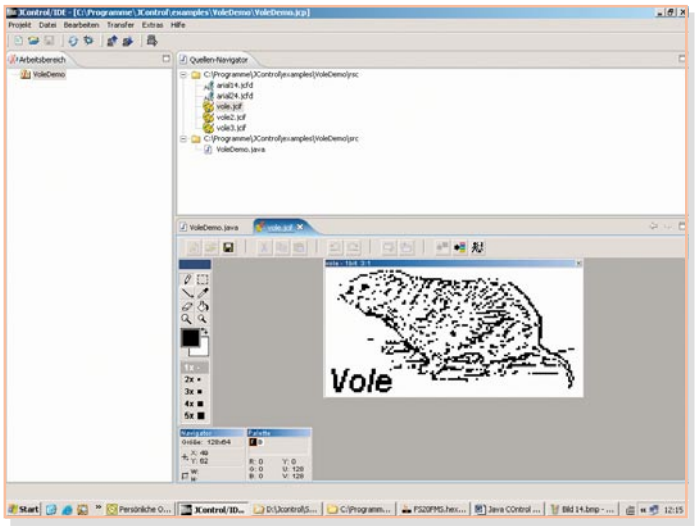


Bild 15: JControl/IDE mit geöffnetem Bildeditor

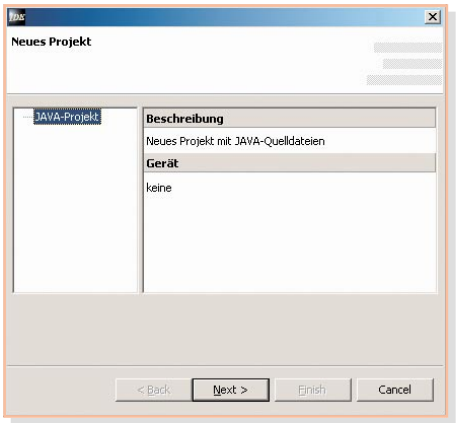


Bild 16: Das Anlegen eines neuen Projektes

Dazu wählen wir zunächst aus dem Menü *„Projekt“* den Menüpunkt *„Neu“* aus. Alternativ kann man auch mit der rechten Maustaste in den Arbeitsbereich klicken und aus dem erscheinenden Popup-Menü gemäß Abbildung 12 den Eintrag *„Neu“* auswählen. Es erscheint ein Fenster, wie in Abbildung 16 zu sehen ist. Hier wird ein Java-

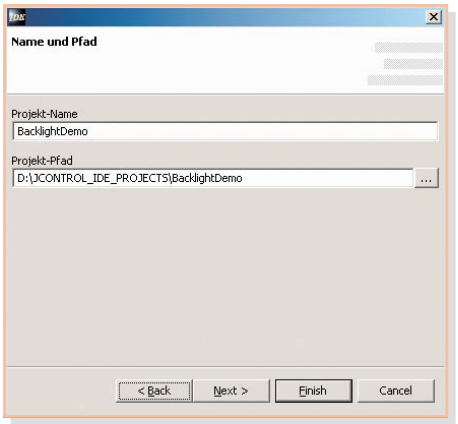


Bild 17: Hier werden Name und Pfad für ein neues Projekt eingegeben.

Projekt erzeugt, mit dem „Next“-Button geht es weiter. Im nächsten Eingabefenster, siehe Abbildung 17, sind der Name des Projektes und der Pfad zu dem Verzeichnis anzugeben, in dem die Projektdatei erstellt werden soll. In unserem Beispiel soll das

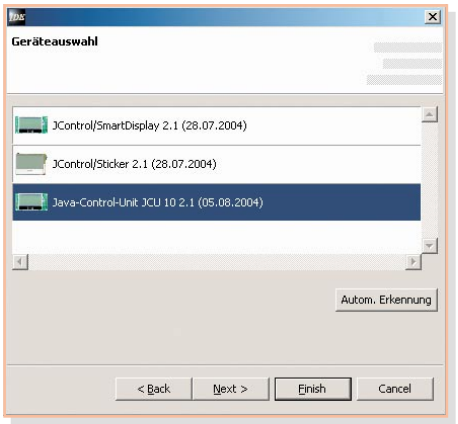


Bild 18: Die Auswahl des Gerätes für ein neues Projekt

Projekt „BacklightDemo“ heißen. Mit dem „Next“-Button gelangt man zur Geräteauswahl, siehe Abbildung 18. Hier wählt man entweder die Java™-Control-Unit mit der entsprechenden Versionsnummer aus oder man versetzt die angeschlossene Java™-Control-Unit in den Download-Mode, wie oben beschrieben, und betätigt im IDE-Programmfenster die Schaltfläche für die automatische Erkennung. Danach kann die Erstellung des neuen Projektes mit dem „Finish“-Button abgeschlossen werden. Das Projekt „BacklightDemo“ erscheint nun im Arbeitsbereich, es ist mit einem Sternchen versehen, weil es noch nicht gespeichert wurde.

Als Nächstes muss eine neue Java-Quelldatei erstellt werden. Hierzu klicken wir mit der rechten Maustaste in den Quellen-Navigator und wählen aus dem Popup-Menü den Menüpunkt *„Neu“* und dann *„Java-Datei“* aus. Daraufhin erscheint ein Eingabedialog (siehe Abbildung 19), in den der Name der Datei eingegeben werden muss. Die Java-Datei nennen wir an dieser Stelle „BacklightDemo“. Außerdem wird das Häkchen vor *„Main-Methode generieren“* gesetzt, da wir eine Hauptklasse erstellen wollen.

Mit einem Doppelklick auf die Datei BacklightDemo.java können wir uns die Datei ansehen. Der Inhalt sollte so aussehen, wie in Abbildung 20 dargestellt.

Als Beispielprogramm wollen wir, wie der Name des Projektes schon ahnen lässt, die Hintergrundbeleuchtung des Displays einstellen und dabei zeigen, wie einfach sich diese Einstellungen mit Komponenten aus dem Vole-Framework realisieren lassen. Das Programm wird einen Slider (Schieberegler) enthalten, mit dem die Hintergrundbeleuchtung in 16 Stufen eingestellt werden kann. Zusätzlich stehen zwei Buttons zur Verfügung, mit denen man die Hintergrundbeleuchtung auf maximale oder minimale Helligkeit schalten kann. Außerdem wird der Name des Demoprogramms im Display angezeigt.

Wir ergänzen jetzt, schrittweise den Erläuterungen folgend, den vorgefertigten Code, bis wir den kompletten Quelltext, wie er in Abbildung 21 zu sehen ist, erstellt haben. Dabei erleichtern die Kommentare das Nachvollziehen der Beschreibung sehr.

Die *„import“*-Anweisungen am Beginn des Quelltextes dienen dazu, die spä-

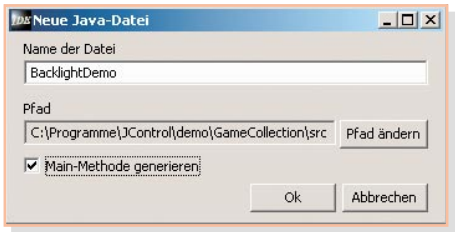


Bild 19: Die neue Java-Quelldatei wird angelegt.

ter verwendeten Klassen aus den verschiedenen Packages bekannt zu machen. Ohne diese Anweisungen würde der Compiler die Klassen nicht finden – dies würde zu entsprechenden Fehlermeldungen führen.

Als Nächstes wird die Definition der Klasse *„BacklightDemo“* um *„extends Frame“* und *„implements ActionListener“* ergänzt. Mit dem Eintrag *„extends Frame“* wird unsere Klasse *„BacklightDemo“* von der Klasse *„Frame“* abgeleitet, d. h. sie erbt alle Variablen und Methoden der Basisklasse und kann um eigene Variablen und Methoden ergänzt werden. Da wir mit „Vole“ arbeiten wollen, ist das Erstellen einer Instanz der Klasse „Frame“ als eine Art Hauptcontainer, in dem andere Komponenten platziert werden können, nötig. Durch das Erben aus der Klasse Frame übernimmt unsere Klasse automatisch die Tastaturabfrage, das Event-Handling und das Fokus-Management.

Die Ergänzung *„implements ActionListener“* implementiert das Interface *„ActionListener“*, das benötigt wird, um ActionEvents von den verwendeten Komponenten zu empfangen.

Bei den verwendeten Komponenten handelt es sich in diesem Programm um ein Label, zwei Buttons und einen Slider. Die Buttons und der Slider lösen beim Betätigen unterschiedliche ActionEvents aus, der ActionListener reagiert auf diese Events und ruft die Methode *„onActionEvent“* auf.

Zunächst müssen wir also unsere Komponenten erstellen, dies geschieht in zwei

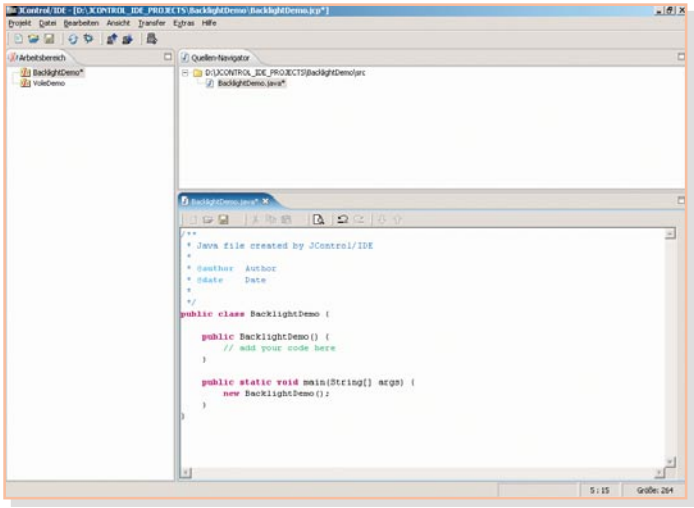


Bild 20: JControl/IDE mit neu angelegter Java-Quelldatei

```
/**
 * Java file created by JControl/IDE
 *
 * @author  Author
 * @date    Date
 * Demoprogramm, bei dem mit zwei Buttons und einem Slider die
 * Hintergrundbeleuchtung eingestellt werden kann
 */
import jcontrol.ui.vole.Frame;
import jcontrol.ui.vole.Label;
import jcontrol.ui.vole.Button;
import jcontrol.ui.vole.Slider;
import jcontrol.ui.vole.event.ActionListener;
import jcontrol.ui.vole.event.ActionEvent;
import jcontrol.io.Backlight;

// Erstellen einer Klasse, die alle Eigenschaften von Frame erbt und
// das Interface ActionListener implementiert
public class BacklightDemo extends Frame implements ActionListener
{
    Label   TEXT_1;                  // Variablen vom jeweiligen Typ anlegen
    Button  B_ON, B_OFF;
    Slider  SLIDE_1;

        public BacklightDemo()       // Konstruktor
    {
        // add your code here
        // Den Variablen ein neues Objekt vom jeweiligen Typen zuweisen
        TEXT_1  = new Label("Backlight Demo", 0, 5, 128, 10, Label.ALIGN_CENTER);
        B_ON    = new Button("Light On", 2, 40, 60, 13);
        B_OFF   = new Button("Light Off", 64, 40, 60, 13);
        SLIDE_1 = new Slider(0,20,128,0,256,16);

        // ActionListerner für die Buttons und den Slider hinzufügen
        // Die ActionListener empfangen ActionEvents von den jeweiligen Komponenten
        B_ON.setActionListener(this);
        B_OFF.setActionListener(this);
        SLIDE_1.setActionListener(this);

        this.add(TEXT_1);            // Label, Buttons und Slider zum Frame hinzufügen
        this.add(B_ON);
        this.add(B_OFF);
        this.add(SLIDE_1);

        this.show();                 // den Frame und seinen Inhalt anzeigen
    }
    // Event Handler, wenn eine Komponente ein ActionEvent auslöst wird
    // diese Methode ausgeführt
    public void onActionEvent(ActionEvent event)
    {   // identifiziere den Button anhand der Ergeignisquelle
        if (event.getSource() == B_ON)
        {// Hintergrundbeleuchtung auf Maximalwert, Slider auf Rechtsanschlag
            Backlight.setBrightness(Backlight.MAX_BRIGHTNESS);
            SLIDE_1.setValue(256);
        }
        // identifiziere den Button anhand der Ergeignisquelle
        else if (event.getSource() == B_OFF)
        {// Hintergrundbeleuchtung aus, Slider auf Linksanschlag
            Backlight.setBrightness(Backlight.MIN_BRIGHTNESS);
            SLIDE_1.setValue(0);
        }
        // identifiziere den Slider anhand der Ergeignisquelle
        else if(event.getSource() == SLIDE_1)
        {// Hintergrundbeleuchtung auf den Wert entsprechend der Sliderposition setzen
            Backlight.setBrightness(SLIDE_1.getValue());
        }
    }
    public static void main(String[] args)
    {
         new BacklightDemo(); // Eine neue Instanz der Klasse BacklightDemo anlegen
    }
}
```

Bild 21: Der Quelltext des Programms „BacklightDemo“

Schritten. Zunächst werden Variablen vom Typ der jeweiligen Klasse deklariert, in unserem Beispiel sind dies die Variablen TEXT_1 vom Typ *„Label“*, B_ON und B_OFF vom Typ *„Button“* und SLIDE_1 vom Typ *„Slider“*. Im zweiten Schritt werden den Variablen mit dem *„new“*-Operator Objekte der entsprechenden Klasse zugewiesen. Die Parameter, die beim Anlegen der neuen Objekte angegeben sind, bestimmen die Eigenschaften dieser Objekte, wie z. B. ihre Beschriftung, ihre Position auf dem Display und ihre Größe. Eine genaue Beschreibung der Parameter findet man in der API-Dokumentation bei den Konstruktoren zu den jeweiligen Klassen.

Hinweis: Einige Klassen sind nicht für alle JControl-Geräte verfügbar. Für welche Geräte eine Klasse verfügbar ist, wird in der Beschreibung dieser Klasse direkt nach dem Autor angegeben. Die Java™-Control-Unit ist hierbei mit dem Smart-Display kompatibel.

An dieser Stelle gehen wir kurz auf die Funktion der Konstruktoren ein, da es in der von uns erstellten Klasse „Backlight-Demo“ als Nächstes auch mit einem solchen Konstruktor weitergeht.

Der Konstruktor ist eine Methode, die den gleichen Namen hat wie die zugehörige Klasse und die automatisch beim Anlegen eines neuen Objektes der Klasse ausgeführt wird. Mit dem Konstruktor können neu angelegte Objekte initialisiert werden. Nehmen wir als Beispiel die Klasse *„Label“*. In der API-Dokumentation kann man sehen, dass diese Klasse drei verschiedene Konstruktoren enthält, die jeweils eine unterschiedliche Anzahl von Parametern haben, mit denen der angezeigte Text, die Position, die Größe und die Ausrichtung festlegbar sind. Beim Anlegen eines neuen Objektes werden diese Parameter in Klammern hinter dem Klassennamen angegeben, und der passende Konstruktor wird ausgeführt.

Der Konstruktor in unserer Klasse unterscheidet sich von denen der Klasse *„Label“* dadurch, dass er keine Parameter hat.

In unserem Konstruktor wird, wie gesagt, der zweite Schritt zum Anlegen der Komponenten durchgeführt. Jeder der vier deklarierten Variablen wird ein neues Objekt zugewiesen, das seine individuellen Eigenschaften durch die angegebenen Parameter erhält.

Damit die Methode *„onActionEvent“* von den beiden Buttons und dem Slider auch aufgerufen wird, übergeben wir mit dem Aufruf der Methoden *„Button/Slider.setActionListener(this)“* einen Zeiger auf die Klasseninstanz und registrieren damit unseren ActionListener. Bei dem *„this“* handelt es sich um einen Zeiger, der auf das aktuelle Objekt zeigt (hier also das Objekt „BacklightDemo“) und automatisch beim Anlegen eines neuen Objektes generiert wird.

In den nächsten Zeilen fügen wir mit der *„add“*-Funktion die neu angelegten Objekte zu unserem „Backlight Demo“ hinzu. Danach wird mit der Methode *„show()“* der Frame sichtbar gemacht, und unser Text, der Slider und die beiden Buttons erscheinen auf dem Display.

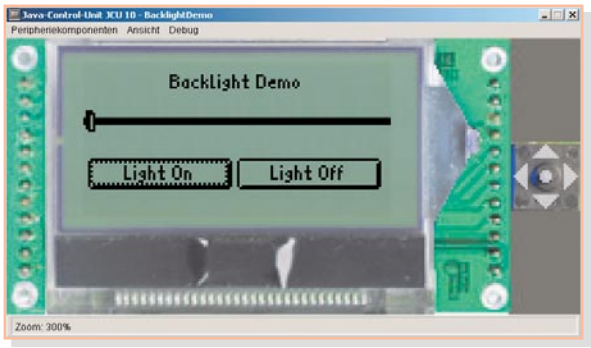


Bild 22: Der Simulator mit dem Programm „BacklightDemo“ in Aktion

Damit ist der Konstruktor abgeschlossen. Wenn man jetzt ein neues Objekt der Klasse „BacklightDemo“ anlegt, werden der Konstruktor ausgeführt, die entsprechenden Objekte im Frame angelegt und sichtbar gemacht und die ActionListener zugewiesen.

Jetzt müssen wir noch die Methode *„onActionEvent“* implementieren, die von den ActionListenern aufgerufen wird. Dies erfolgt in den nächsten Zeilen des Quelltextes.

Wenn die Methode *„onActionEvent“* vom ActionListener aufgerufen wurde, ist ein bestimmtes Ereignis eingetreten. In unserem Beispiel gibt es nur drei Objekte, die ein Event auslösen können, entweder wurde einer der beiden Buttons betätigt, oder der Slider wurde verschoben. Herauszufinden, welches Objekt die Ursache für das Event war, und eine entsprechende Reaktion darauf zu veranlassen, ist nun wieder unsere programmiertechnische Aufgabe.

Vom ActionListener bekommen wir beim Aufruf von *„onActionEvent“* ein Objekt vom Typ *„ActionEvent“* als Parameter geliefert. Die Methode *„getSource()“* liefert uns den Namen des Objektes, welches das Event ausgelöst hat. Jetzt brauchen wir nur noch zu vergleichen, ob das Event von B_ON, B_OFF oder SLIDE_1 ausgelöst wurde, und eine entsprechende Aktion starten.

Die Aktionen der beiden Buttons sind nahezu identisch, hier wird in beiden Fällen mit der Methode *„setBrightness(int)“* die Hintergrundbeleuchtung entweder auf 0 (*MIN_BRIGHTNESS*) oder auf 256 (*MAX_BRIGHTNESS*) gesetzt. Entsprechend dazu erfolgt mit der Methode *„setValue(int)“* aus der Klasse *„Slider“* das Ändern der Position des Sliders entweder auf 0 (Linksanschlag) oder 256 (von uns beim Anlegen des Objektes definierter Rechtsanschlag).

Beim Slider ist der aktuelle Wert abzufragen, auf den die Hintergrundbeleuchtung gesetzt werden soll. Da wir den Slider mit einem Wertebereich von 0 bis 256 definiert haben, passt der von der Methode *„getValue()“* aus der Klasse *„Slider“* gelieferte Wert genau zu den Werten für die Methode *„setBrightness(int)“* aus der Klasse *„Backlight“*. So kann man die Methode *„getValue()“* direkt als Parameter in *„setBrightness(int)“* einsetzen.

Bleibt als Letztes, noch eine Instanz der Klasse „BacklightDemo“ anzulegen, damit wir auch die gewünschte Funktion erhalten. Dies geschieht in der *„Main-Methode“*, die ja bereits beim Anlegen der Java-Datei von der JControl/IDE angelegt wurde. Dort sehen wir bereits den Eintrag *„new BacklightDemo()“*. Also ist unser Programm jetzt fertig und sollte bei einem Klick auf das Disketten-Symbol (speichern) ohne Fehlermeldungen kompiliert werden.

Programmsimulation

An dieser Stelle können wir auch das Programm zunächst mit dem Simulator testen, bevor wir es auf die Java™-Control-Unit laden. Dazu klicken wir einmal auf das Simulatorsymbol ganz rechts in der Symbolleiste. Der Simulator startet, und es öffnet sich ein neues Fenster, wie in Abbildung 22 zu sehen. Die Taster, die neben dem Display dargestellt sind, haben die gleichen Funktionen wie die auf dem Evaluation-Board und können mit der Maus betätigt werden.

Die Tastaturabfrage und das Fokus-Management werden ja, wie bereits gesagt, dadurch, dass unsere Klasse die Klasse „Frame“ erweitert, im Hintergrund abgearbeitet.

Für die Bedienung bedeutet dies, dass wir mit der „Left“- und der „Right“-Taste den Fokus weitergeben, d. h. das aktive Objekt wechseln können. In unserem Demoprogramm hat der linke Button nach dem Einschalten den Fokus, dies ist durch einen gestrichelten Rahmen innerhalb des Buttons erkennbar. Hat ein Button den Fokus, kann er mit der „Select“-Taste betätigt werden. Erhält der Slider den Fokus, wird dies durch einen massiven Rahmen angezeigt, mit der „Up“- und der „Down“-Taste ändert man dann die Position des Sliders.

Das selbst geschriebene Projekt kann nun, wie schon bei den fertigen Beispielprogrammen beschrieben, in den Speicher der Java™-Control-Unit geladen und dort über die Bedienung der Tasten ausgeführt werden.

In der nächsten Ausgabe werden wir, anhand von Beispielen, Tipps und Hilfestellungen zum Erstellen eigener Applikationen für die Java™-Control-Unit geben.