JAVA HYPERTOOL

Training Manual

November 2016

Document:

Java HyperTool Training Manual rev.2.0
November 2016

Copyright:

Trademarks:

All product names cites are trade names or registered trademarks of their respective owners.

For more information, contact:

Ucamco NV

Bijenstraat 19

B-9051 Gent

Belgium

✆ +32 9 216 99 00

🖷 +32 9 216 99 12

Email: hypertool@ucamco.com

Web Site: http://www.ucamco.com

Table of Contents

Contents

Java HyperTool Training Manual

Overview

Introduction

The following manual contains the material presented during the Java HyperTool Training.

Contents

This book contains the following chapters.

Chapter 1 – Getting Started

Overview

Introduction

This chapter explains the required setup to follow the exercises and how to write a basic Java program.

Contents

This chapter contains the following topics.

Required Setup

UCAM installation

The training requires UCAM v6.1-2 or higher being installed.

JDK

UCAM v6 comes with a Java Runtime Environment. This does not allow compiling Java code. The training requires Java Developers Kit 1.2.2 or higher being installed (for UCAM v7 a Java Developers Kit 1.4.x or higher is required).

Implementations can be downloaded from *http://java.sun.com*.

Documentation

Training attendees will receive:

- All the Java source code used during this training in the sources directory
- This manual in pdf format
- The Java HyperTool API specification in JavaDoc format : 'API reference/index.html'

A comprehensive tutorial on Java can be found at *http://www.javasoft.com/docs/books/tutorial/index.html*.

Conventions

- When no distinction is made between Windows and Unix platforms, a forward slash (/) will be used as directory separator. For Windows platforms replace it with a backslash (\).
- The HOME directory is used to store custom UCAM files.
  - For Unix: The HOME directory is defined in the *.cshrc* file and defaults to the login directory of the current user.
  - For Windows: By default, there is no HOME environment variable defined. When UCAM does not find a HOME environment variable, it defaults to *%USERPROFILE%\ucam\home*. For the purpose of the training, it is advised to define the HOME variable as the directory where all custom files will be stored.
- The ETSCAM_INSTALL directory is the directory where UCAM is installed. On Unix systems, this environment variable is set in the *.cshrc* file during installation. On Windows, this variable is set in the *ucam_env.bat* file. For the purpose of the training, it is advisable to define this variable for the terminal window used for Java compilation.
- The commands for compiling and running Java examples require that the *bin* directory of the JDK installation is included in the *PATH*. If not, the full path for *javac* and *java* should be given.

Writing a basic Java program

Task

Create a simple method that prints out “Hello World !” on the terminal window.

The code

```
/*
 * HelloWorld.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

/**
 * Basic Java application. Prints out "Hello World !".
 */
public class HelloWorld {
  // Main method required to start up as a standalone application.
  // Calls the sayHello() method.

   public static void main(String[] args) {
     sayHello();
   }

  // Functional block for the application.
  // Prints out "Hello World !".

   public static void sayHello() {
     System.out.println("Hello World !");
   }
}
```

Store the file as ***sources/HelloWorld.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix :

  `javac –d $HOME sources/HelloWorld.java`

- For Windows :

  `javac –d %HOME% sources\HelloWorld.java`

A class *HelloWorld.class* is now created in the HOME directory.

Continued on next page

Writing a basic Java program, ***Continued***

Run the application

From the HOME directory start the Java virtual machine with the resulting class:

```
java HelloWorld
```

The output *Hello World !* appears in the terminal window.

Java Packages

Introduction

Packages can be used to organize your classes into functional units. They correspond to **a directory structure the Java Virtual Machine uses to search for classes.**

As a general rule, one should start a package name with the reverse of the company URL, in order to avoid name clashes with packages from other companies/divisions.

EXAMPLE: **package com.company.ucam.hello**

The resulting class file can be found in *com/company/ucam/hello*.

Classes outside this package can use the given class using import statements.

EXAMPLE: **import com.company.ucam.hello**

The ***classpath*** setting defines where the **Java compiler (*javac*) or the Java Virtual Machine (*java*) should look for the class files**.

Task

Adapt the HelloWorld application to reside in the package com.company.ucam.hello.

The code

```
/*
 * HelloWorld.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.hello;

/**
 * Basic Java application. Prints out "Hello World !".
 */
public class HelloWorld {
  // Main method required to start up as a standalone application.
  // Calls the sayHello() method.
  public static void main(String[] args) {
    sayHello();
  }
```

Continued on next page

Java Packages, ***Continued***

The code *(continued)*

```
  // Functional block for the application.
  // Prints out "Hello World !".
  public static void sayHello() {
    System.out.println("Hello World !");
  }
}
```

Store the file as ***sources/com/company/ucam/hello/HelloWorld.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix :

```
javac –d $HOME
sources/com/company/ucam/hello/HelloWorld.java
```

- For Windows:

```
javac –d %HOME%
sources\com\company\ucam\hello\HelloWorld.java
```

A class ***com/company//ucam/hello/HelloWorld.class*** is now created in the HOME directory.

Run the application

From the HOME directory, start the Java virtual machine with the resulting class.

- For Unix :

```
java –classpath $HOME com.company.ucam.hello.HelloWorld
```

- For Windows :

```
java –classpath %HOME% com.company.ucam.hello.HelloWorld
```

The output *Hello World !* appears in the terminal window.

Chapter 2 - Customizing UCAM

Overview

Introduction

This chapter explains how to add functionality to the UCAM menubars and toolbars.

Contents

This chapter contains the following topics.

Adding functionality to UCAM

Introduction

There are different ways to add custom programs to UCAM. The UCAM menu and the toolbar can be configured using a resource file. New functionality can be added using the ***UcamMenuActions*** and ***UcamActions*** classes.

The UcamMenuActions class

One can create its own ***UcamMenuActions*** class with custom code included in a *static* block.

The class is loaded before the menus and toolbars are built. This allows defining functionality, which should be included in a menu and/or toolbar.

At startup, **UCAM checks its *classpath* for a class named *UcamMenuActions*** in the default package (root package/no package defined). The order in which is searched is:

- The user HOME directory
- The ETSCAM_CFG directory
- The ETSCAM_DAT directory

When a class is found, it is loaded, causing its static block to be executed.

The Umnbutton and Umntbutton classes

Umnbutton and Umntbutton instances define the behaviour of a Pushbutton/Togglebutton without the buttons itself being created. They serve as the template for the buttons to be added to a menu and/or toolbar.

See the API documentation for the description of the classes.

Customizing the main menu

The **UCAM main menu can be customized using the resource file *umainmenu***.

UCAM looks for the file at the following locations:

- HOME : user defaults
- ETSCAM_CFG : site defaults
- Barco defaults

The *menubar* property defines a list of all menus. Each menu(item) can further define a list of menu items.

A separator is represented by a hyphen (-) and is only placed when there are menu items before and after the separator.

A menu item is only created when a Umnbutton or Umntbutton instance with the same name exists.

A menu item can have the following properties:

- *item.name* : the label for the button
- *item.enabled* : true or false
- *item.state* : 1 or 0 (only for Umntbuttons)

Customizable menus

- *umainmenu* : the main menu
- *ujobmenu* : the job buildup menu in job mode
- *ublomenu* : the job buildup menu in block mode
- *uerrmenu* : the error viewer menu

Customizing toolbars

UCAM has a built in toolbar editor to edit the toolbars. One can however edit the toolbar resource files manually.

The place where these files are stored is defined in the ***toolbar.setup.file*** property in ***ucam.db***.

The *items* property defines a list of all the buttons.

A toolbar item can have the following properties:

- *item.icon* : The icon file (.gif)
- *item.tooltip* : The button tooltip

The UcamActions class

One can create its own ***UcamActions*** class with custom code included in a *static* block.

The **class is loaded after the menus and toolbars are built**. This allows defining functionality, which can be added at the end of an existing menu.

At startup, **UCAM checks its *classpath* for a class named *UcamActions*** in the default package (root package/no package defined). The order in which is searched is:

- The user HOME directory
- The ETSCAM_CFG directory
- The ETSCAM_DAT directory

When a class is found, it is loaded, causing its static block to be executed.

The add_pb method

The ***Ucamapp*** class has a class method ***add_pb()*** which allows to directly add a menu item to the end of a menu.

The method can be accessed as ***Ucamapp.cO.add_pb()***.

See the API documentation for the description of the method.

Customizing the Menu

Task

Add the *Hello World* functionality to a new item in the *HyperTool* menu, using the UcamMenuActions class.

The code

```
/*
 * UcamMenuActions.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the HelloWorld class.
import com.company.ucam.hello.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Class loaded by Ucam.
 * Only its static block is executed.
 */
public class UcamMenuActions {

  // Static block executed when the class is loaded.
   static {

    // Action to associate with the Umnbutton.
    // Its action() method is executed when the button
    // is clicked.
    // Calls the sayHello() method from the
    // com.company.ucam.hello.HelloWorld class.

     Ucamaction act = new Ucamaction() {
       public void action() {
         HelloWorld.sayHello();
       }
     };
```

Continued on next page

Customizing the Menu, ***Continued***

The code *(continued)*

```
    // Create a new Umnbutton named 'Hello'.
    // The name can be used in the menu and toolbar resource files.

     new Umnbutton("Hello", act);
   }
}
```

Store the file as ***sources/UcamMenuActions.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix :

```
javac -d $HOME -classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/UcamMenuActions.java
```

- For Windows :

```
javac -d %HOME% -classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\UcamMenuActions.java
```

A class *UcamMenuActions.class* is now created in the HOME directory.

The umainmenu resource file

Copy the *umainmenu* file from *ETSCAM_CFG* or *ETSCAM_INSTALL/classes/resources* to the HOME directory if it is not there yet.

In the *umainmenu* file, edit the line defining the HyperTool menu as follows:

```
hypertool_menu : hypertool_grab Hello
```

Run the application

Start UCAM.

A *Hello* item was added to the *Hypertool* menu.

Select the *Hello* item.

The output *Hello World !* appears in the terminal window.

Task

Add the *Hello World* functionality to the SmartStart button called tb_job_input.

The code

```
/*
* UcamMenuActions.java
*
* Copyright (c) 2016 Ucamco All rights reserved.
*/
// Allows to locate the HelloWorld class.
import com.company.ucam.hello.*;
// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;
import java.awt.event.* ;
/**
* Class loaded by Ucam.
* Only its static block is executed.
*/
public class UcamMenuActions {
// Static block executed when the class is loaded.
static {
ActionListener myAction = new ActionListener() {
public void actionPerformed(ActionEvent e) {
HelloWorld.sayHello();
}
};
Ucamapp.cO.addCustomAction("tb_job_input", myAction);
}
}
```

Store the file as ***sources/UcamMenuActions.java*** in the HOME directory.

Continued on next page

,
Continued

Compile the code

From the HOME directory, run the Java compiler.

- For Unix :

```
javac -d $HOME -classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/UcamMenuActions.java
```

- For Windows :

```
javac -d %HOME% -classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\UcamMenuActions.java
```

A class *UcamMenuActions.class* is now created in the HOME directory.

Run the application

Start UCAM.
Push the SmartStart button and the output *Hello World !* appears in the terminal window.

Adding items to a menu

Task

Add the *Hello World* functionality to a new item in the 'HyperTool' menu, using the UcamActions class.

The code

```
/*
 * UcamActions.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the HelloWorld class.
import com.company.ucam.hello.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Class loaded by Ucam.
 * Only its static block is executed.
 */
public class UcamActions {

  // Static block executed when the class is loaded.
  static {

    // Action to associate with the menu item.
    // Its action() method is executed when the button
    // is clicked.
    // Calls the sayHello() method from the
    // com.company.ucam.hello.HelloWorld class.
    Ucamaction act = new Ucamaction() {
      public void action() {
        HelloWorld.sayHello();
      }
    };
```

Continued on next page

Adding items to a menu, ***Continued***

The code *(continued)*

```
    // Adds a button called 'say_hello' with label 'Say Hello'
    // to the hypertool menu.

     Ucamapp.cO.add_pb("say_hello",
       "Say Hello", null, act,
       "hypertool_menu", true);
  }
}
```

Store the file as ***sources/UcamActions.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac –d $HOME –classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/UcamActions.java
```

- For Windows:

```
javac –d %HOME% –classpath
%HOME%;%$ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\UcamActions.java
```

A class *UcamActions.class* is now created in the HOME directory.

Run the application

Start UCAM.

A *Say Hello* item was added to the *Hypertool* menu.

Select the *Say Hello* item.

The output *Hello World !* appears in the terminal window.

Chapter 3 – Designing User Interfaces

Overview

Introduction This chapter explains how to design user interfaces and add them to Ucam.

Contents This chapter contains the following topics.

HyperTool User Interface APIs

Swing

Swing is a standard Java User Interface package included in Java 2. It contains an extensive set of UI elements. The basic package for Swing is *javax.swing*.

An extensive tutorial about Swing can be found at ***http://java.sun.com/docs/books/tutorial/index.html***.

The HyperTool ui package

The previous versions of UCAM (non-Java based) contained a HyperTool API for User Interfaces based on X Windows/Motif. This API has been converted to Java and published as the *com.barco.ets.ucam.ui* package.

This package is primarily available to support automatic translation of old HyperTool code to Java based code. Its use for new developments is highly discouraged.

Mixing Swing with the ui package

The **HyperTool ui package is based on Swing**, which allows existing User Interfaces written with the ui package to be extended using Swing.

Each component from the ui package has an embedded Swing component: the *javaPeer*. This Swing component can be retrieved using the *getJavaPeer()* method from the *com.barco.ets.ucam.ui.Uiobj* class.

Adding windows to UCAM

Except for the UCAM main window, all **UCAM windows are based on** Swing ***JDialog*** instances attached to the main window. The *Uiobj.getFrame()* class method allows to retrieve the *JFrame* component of the UCAM main window which allows to add new *JDialog* instances to UCAM.

Writing an attribute viewer

Task

Write an application, which displays a window containing a table with all the attributes of the current Job.

Add two buttons:

- Refresh to update the attribute table
- Cancel to close the window

The functionality should be callable from a menu item in the HyperTool menu.

The code

```
/*
 * JobAttributeLister.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.attributes;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

// Additional User Interface packages needed.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import javax.swing.table.*;
import javax.swing.event.*;
```

Continued on next page

Writing an attribute viewer, ***Continued***

The code *(continued)*

```
/**
 * Implements a JDialog which displays all the attributes
 * of the current job in a JTable.
 */
public class JobAttributeLister extends JDialog {

// Only one dialog can be created. Once created, it is
  // recycled for later use.
  private static JobAttributeLister dialog = null;

  // The model used to store the attribute data in.
  private DefaultTableModel model;

  // Constructs a new dialog window and initializes all
  // values.
  public JobAttributeLister() {

    // Creates and initializes the JDialog with the
    // Ucam main window as parent and 'Job Attributes'
    // as title.
     super(Uiobj.getFrame(), "Job Attributes");
     getContentPane().setLayout(new BorderLayout());
     ((JPanel)getContentPane()).setBorder(new EmptyBorder(5,
       5, 5, 5));

    // The data for the table is stored in the model,
    // which is initialized with 2 columns.
     model = new DefaultTableModel(new Object[0][0], new
       Object[] {"Name", "Value"});
     JTable table = new JTable(model);
     table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

    // Add the table to the center of the dialog.
    // The table will resize together with the dialog.
     getContentPane().add(new JScrollPane(table),
       BorderLayout.CENTER);
```

Continued on next page

Writing an attribute viewer, ***Continued***

The code *(continued)*

```
    // Create a panel to display the control buttons.
    JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 5,
      5));
    buttonPanel.setBorder(new EmptyBorder(5, 0, 0, 0));

    // The refresh button.
    JButton button = new JButton("Refresh");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        refresh();
      }
    });
    buttonPanel.add(button);

    // The cancel button.
    button = new JButton("Cancel");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        dialog.hide();
      }
    });
    buttonPanel.add(button);

    getContentPane().add(buttonPanel, BorderLayout.SOUTH);

    // Resize the dialog to its preferred size.
    pack();
  }
  /**
   * Reloads the attributes from the current job and
   * stores them in the table.
   * Called when opening the dialog or pressing the
   * refresh button.
   */
```

Continued on next page

Writing an attribute viewer, ***Continued***

The code *(continued)*

```
public void refresh() {

  // Do nothing when there is no current job.
   if (Ucamv6.ucam_job == null) {
     return;
   }

  // Retrieve all the attributes.
   Uobjattrlist attrList = Ucamv6.ucam_job.attributes();
   Uattrobj attr;
   Object[][] values = new Object[attrList.used()][2];


   for (int i = 1; i <= attrList.used(); ++i) {
     attr = (Uattrobj)attrList.at(i);
     if (attr != null) {
       values[i - 1][0] = attr.name();
       values[i - 1][1] = attr.value();
     }
   }

  // Update the model.
   model.setDataVector(values, new Object[] {"Name",
     "Value"});
}

/**
 * Refreshes and shows the window.
 * Called when the menu item is selected.
 */
public static void showDialog() {

  // Create a new dialog when needed.
   if (dialog == null) {
     dialog = new JobAttributeLister();
   }
```

Continued on next page

Writing an attribute viewer, ***Continued***

The code *(continued)*

```
    dialog.refresh();
    dialog.show();
  }
}
```

Store the file as ***sources/com/company/ucam/attributes/JobAttributeLister.java*** in the HOME directory.

```
/*
 * UcamActions.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the JobAttributeLister class.
import com.company.ucam.attributes.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Class loaded by Ucam.
 * Only its static block is executed.
 */
public class UcamActions {

  // Static block executed when the class is loaded.
  static {

    // Action to associate with the menu item.
    // Its action() method is executed when the button is clicked.
    // Calls the showDialog() method from the
    // com.company.ucam.attributes.JobAttributeLister class.
```

Continued on next page

Writing an attribute viewer, ***Continued***

The code *(continued)*

```
    Ucamaction act = new Ucamaction() {
      public void action() {
        JobAttributeLister.showDialog();
      }
    };

   // Adds a button called 'show_attributes' with label
   // 'Show Attributes' to the hypertool menu.
    Ucamapp.cO.add_pb("show_attributes",
      "Show Attributes", null, act,
      "hypertool_menu", false);
  }
}
```

Store the file as ***sources/UcamActions.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac -d $HOME -classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/attributes/JobAttributeLister.java
sources/UcamActions.java
```

- For Windows:

```
javac -d %HOME% -classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\attributes\JobAttributeLister.java
sources\UcamActions.java
```

The *classes com/company/ucam/attributes/JobAttributeLister.class* and *UcamActions.class* are now created in the HOME directory.

Run the application

Start UCAM.

A *Show Attributes* item was added to the *Hypertool* menu.

Open a job.

Select the *Show Attributes* item.

A new window pops up showing the attributes for the job.

Chapter 4 – The HyperTool package

Overview

Introduction

This chapter describes the **major classes** in the HyperTool package and how to use them. A number of exercises demonstrate their practical use.

All classes from the HyperTool package can be found in *com.barco.ets.ucam.hypertool*.

Contents

This chapter contains the following topics.

Dtl$object

Description

In order **to minimize the effort to convert existing HyperTool** scripts from UCAM v5 and earlier to the new Java based HyperTool, **the existing HyperTool API has been ported to a Java API** which matches as good as possible.

The Class Object

In order **to mimic the existing HyperTool functionality in Java**, the class methods from the HyperTool classes could not be implemented as class methods in Java, but instead **the notion of a *Class Object* had to be introduced**.

Each class in the Hypertool API descends from the *Dtl$object* class and contains all the instance methods of the original HyperTool class. **Each of the classes has a class variable *cO,* which can be used to access the class method of the original HyperTool class as instance methods**. The class of the *cO* variable is an inner class called *CO*.

EXAMPLE: The class methods of the *Ucamapp* class can be accessed through *Ucamapp.cO*.

Future developments

The currently published API is considered frozen and uses the above-described mechanism.

Future developments might rely on normal Java class and instance methods. It is therefor advised to consult the API documentation for details.

Ucamobj and subclasses

Description

The *Ucamobj* class is the common superclass of a number of important UCAM classes. It defines the common behavior available in each of these subclasses.

The subclasses are:

- *Ujob*
- *Ulayer*
- *Ucore*
- *Uape*

Ujob

The *Ujob* class represents a job in UCAM. A job contains a number of layers (*Ulayer* or *Ucore* instances). It also holds information about the job name, fixture type and drc parameters.

Layer classes

The HyperTool package contains classes to represent the different layer types:

- *Ucore* : represents a core
- *Ulayer* : superclass for the different layer types :
  - *Udrilayer* : represents a drill layer
  - *Uextlayer* : represents an extra layer (mask, silk, etc.)
  - *Usiglayer* : represents a signal layer

Layer instances contain a list of apertures (*Uape* instances) and additional information like name, reference points, etc.

Aperture classes

The different aperture types are represented by subclasses of the *Uape* class:

- *Ubloape* : block apertures
- *Uboxape* : box apertures
  - *Uoctape* : octagonal box apertures
- *Ucirape* : circular apertures
- *Ucomape* : complex apertures
  - *Utheape* : thermal apertures
- *Uconape* : contour apertures
- *Udonape* : donut apertures
- *Urecape* : rectangular apertures
- *Utxtape* : text apertures

The Ucamapp class

Description

The *Ucamapp* class defines the basic UCAM functionality in its class methods.

Subclassing Ucamapp

UCAM accesses the *Ucamapp* functionality through the *Ucamv6.ucam$app* variable. The variable is by default set to *Ucamapp.cO*. One can subclass *Ucamapp* and overload the methods in *Ucamapp$CO* to change the default behavior. This change is effected by setting *Ucamv6.ucam$app* to the Class Object of the *Ucamapp* subclass.

It is therefor advised to access *Ucamapp* methods through the *Ucamv6.ucam$app* variable.

Subclassing Ujob

In order to change the default behavior of the job loaded in UCAM, create a subclass of *Uxjob* with the necessary overloaded methods.

Next, create a subclass of *Ucamapp* and overload the *jobclass()* method to return the Class Object of the custom job class. Set the *Ucamv6.ucam$app* variable to the Class Object of the custom *Ucamapp* class.

When a job is opened, or a new job is created, it will be an instance of the custom job class.

Drawing layers

Introduction

UCAM contains a highly optimized drawing engine to generate accurate images of layers, and combine these layers to present a comprehensible image of all selected layers within a job.

The Displaypar class

The *Displaypar* class represents an off-screen drawing area for UCAM objects.

The viewport in combination with the *Displaypar* size represents the current transformation between world coordinates (the actual Job coordinates in the currently set unit) and screen coordinates (the pixels on the screen).

A layer can be drawn in a *Displaypar* instance using the *display()* method in the *Ulayer* class.

The Udwaplane class

The *Udwaplane* class is a specialized subclass of *Drawingarea*, which handles a set of *Displaypar* and *Uplane* instances.

The *Displaypar* instances correspond with the UCAM planes as defined in the Job Buildup window and have their corresponding color.

The *Uplane* instances allow adding additional drawing using the *Uplane* draw methods. All drawing operations appear on an off-screen buffer which is copied to the screen after a *repaint()* call. When the contents of the *Displaypar* instances needs to be displayed on the screen, the *updateBitmap()* method needs to be called before the *repaint()* call.

Write a layer viewer

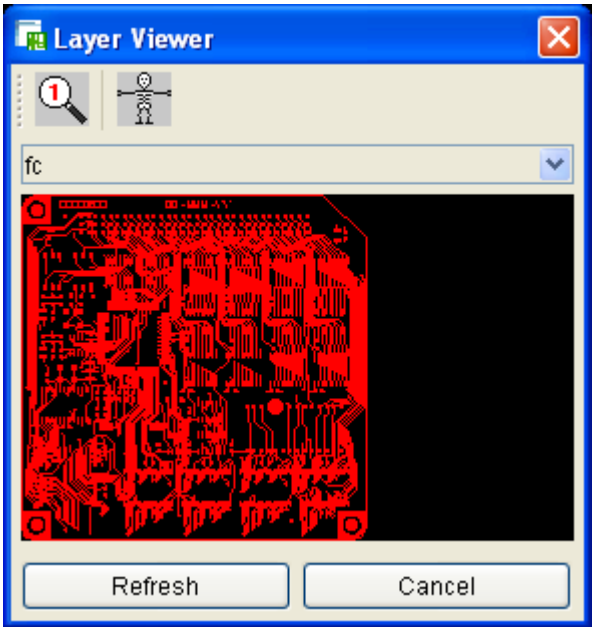
Task

Write an application, which displays one layer at a time.

The window should contain the following components:

- A drawing area to display the layer
- A combobox listing all the layers in the current job
- A button to restore the view to total
- A togglebutton to switch the skeleton setting
- A refresh button to refresh the combobox
- A cancel button to close the window

The functionality should be callable from a menu item in the HyperTool menu.



The code

```
/*
 * LayerViewer.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.layer;
```

```
// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

// Additional User Interface packages needed.
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
```

Continued on next page

Write a layer viewer, ***Continued***

The code *(continued)*

```
/**
 * Implements a JDialog with an Udwaplane to display
 * a layer selected in a JComboBox.
 */
public class LayerViewer extends JDialog {

  // Only one dialog can be created. Once created, it is
  // recycled for later use.
  private static LayerViewer dialog = null;

  // The combobox containing the names of all layers.
  private JComboBox comboBox;

  // The drawing area.
  private Udwaplane dwa;

  // The javaPeer of the drawing area.
  private JPanel panel;

  // The DisplayPar of plane 1 of the drawing area.
  private Displaypar dsp;

  // The Graphics to draw directly to the screen.
  private Graphics panelGraphics;

  // The name of the currently selected layer.
  private String layerName;

  // The toggle for the skeleton setting.
  private JToggleButton skeletonButton;

  // The last dragged rectangle.
  private int[] dragRect;
```

Continued on next page

Write a layer viewer, ***Continued***

The code *(continued)*

```
/**
 * Constructs a new dialog window and initializes all
 * values.
 */
public LayerViewer() {

  // Creates and initializes the JDialog with the
  // Ucam main window as parent and 'Layer Viewer'
  // as title.
   super(Uiobj.getFrame(), "Layer Viewer");
   getContentPane().setLayout(new BorderLayout());

   JPanel interior = new JPanel(new BorderLayout(5, 5));
   interior.setBorder(new EmptyBorder(5, 5, 5, 5));
   ((JPanel)getContentPane()).add(interior,
     BorderLayout.CENTER);

  // The changeLayer() method is called when an item
  // is selected in the layer combobox.
   comboBox = new JComboBox();
   comboBox.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
       changeLayer();
     }
   });
   interior.add(comboBox, BorderLayout.NORTH);

  // The changeLayer() method is called when the drawing
  // is resized, or a repaint is requested by the OS.
   dwa = new Udwaplane(null, "layer_viewer_dwa");
   Ucamaction act = new Ucamaction() {
     public void action() {
       changeLayer();
     }
   };
```

Continued on next page

The code *(continued)*

```
dwa.setpaintaction(act);
dwa.setresizeaction(act);

// The dragRectangle() method is called when the user
// moves the mouse over the drawing area with a mouse
// button held down.
dwa.setmoveaction(new Ucamaction() {
  public void action() {
    dragRectangle();
  }
});

// The zoom() method is called when the user releases
// the mouse button over the drawing area.
dwa.setreleaseaction(new Ucamaction() {
  public void action() {
    zoom();
  }
});

panel = (JPanel)dwa.getJavaPeer();
panel.setPreferredSize(new Dimension(320, 240));
interior.add(panel, BorderLayout.CENTER);

// Create a panel to display the control buttons.
JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 5,
  5));
buttonPanel.setBorder(new EmptyBorder(5, 0, 0, 0));

// The refresh button.
JButton button = new JButton("Refresh");
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    refresh();
  }
});
```

Continued on next page

The code *(continued)*

```
 buttonPanel.add(button);

// The cancel button.
 button = new JButton("Cancel");
 button.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     dialog.hide();
   }
 });
 buttonPanel.add(button);
 interior.add(buttonPanel, BorderLayout.SOUTH);

// Create a toolbar for the totalView and skeleton buttons.
 JToolBar toolBar = new JToolBar();

// The totalView button.
 button = new JButton(new ImageIcon
   (getClass().getResource("/icons/nozoom.gif")));

 button.setMargin(new Insets(0, 0, 0, 0));
 button.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     dialog.totalView();
   }
 });
 toolBar.add(button);

 toolBar.addSeparator();

// The skeleton button.
 skeletonButton = new JToggleButton(new ImageIcon
   (getClass().getResource("/icons/skeleton0.gif")));

 skeletonButton.setMargin(new Insets(0, 0, 0, 0));
```

Continued on next page

The code *(continued)*

```
    skeletonButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        dialog.skeleton();
      }
    });
    skeletonButton.setSelectedIcon(new ImageIcon
      (getClass().getResource("/icons/skeleton1.gif")));

    toolBar.add(skeletonButton);

    getContentPane().add(toolBar, BorderLayout.NORTH);

   // Resize the dialog to its preferred size.
    pack();
  }

 /**
  * Checks which layer is selected, loads the layer
  * and displays it. When the layer is changed, a total
  * view is displayed.
  *
  * Called when an item is selected in the layer combobox,
  * or when anything needs to be repainted.
  */
 public void changeLayer() {

   // Check if there is a job loaded.
    if (Ucamv6.ucam_job == null) {
      return;
    }

   // Check if the drawing area is already initialized.
    dsp = dwa.dsp[1];
    if (dsp == null) {
      return;
    }
```

Continued on next page

The code *(continued)*

```
// Initialize the Graphics object for direct drawing
// to the screen.
 panelGraphics = panel.getGraphics();
 panelGraphics.setXORMode(Color.white);

// Get the currently selected layer.
 String name = (String)comboBox.getSelectedItem();

 Ulayer layer = Ucamv6.ucam_job.getlayerbyname(name);
 if (layer == null) {
   return;
 }

 layer.load();

// If a new layer is selected, initialize the dsp
// to display a total view.
 if (!name.equals(layerName)) {
   layerName = name;
   dsp.totalview(layer.enclosingbox());
 }

// Draw the layer in the dsp.
 layer.display(dsp, null);

// Clear the previous rubber rectangle.
 dragRect = null;

// Draw to the screen.
 dwa.updateBitmap();
 dwa.repaint();
}
```

Continued on next page

Write a layer viewer, ***Continued***

The code *(continued)*

```
/**
 * Reloads the layer names from the current job and puts
 * them in the combobox.
 * Select the first item.
 *
 * Called when opening the dialog or pressing the
 * refresh button.
 */
public void refresh() {
  if (Ucamv6.ucam_job == null) {
    return;
  }

 // Clear the combobox.
  comboBox.removeAllItems();

 // Fill up the combobox.
  for (int i = 1; i <= Ucamv6.ucam_job.numlayers(); ++i) {
    comboBox.addItem(Ucamv6.ucam_job.getlayer("all", null,
      i).name());
  }

 // Select the first item, resulting in a totalview
 // of the first layer.
  comboBox.setSelectedIndex(0);
}

/**
 * Implements rubberbanding functionality.
 *
 * Called when moving the mouse over the drawing area with
 * a button held down.
 */
```

Continued on next page

The code *(continued)*

```
public void dragRectangle() {
  if (panelGraphics == null) {
    return;
  }

 // We are not interested in the right mouse button.
  if (Uiobj.cO.state.getInt("mb") == 4) {
    return;
  }

  int x = Uiobj.cO.state.getInt("x");
  int y = Uiobj.cO.state.getInt("y");
  int pressX = Uiobj.cO.state.getInt("pressx");
  int pressY = Uiobj.cO.state.getInt("pressy");

 // Erase the old rectangle when there was one.
 // Erasing is done by drawing the rectangle a second
 // time. The XOR setting in panelGraphics makes
 // sure this erases the previous rectangle.
  if (dragRect != null) {
    panelGraphics.drawRect(dragRect[0], dragRect[1],
      dragRect[2], dragRect[3]);
  }

  dragRect = new int[4];
  if (x > pressX) {
    dragRect[0] = pressX;
    dragRect[2] = x - pressX;
  }
  else {
    dragRect[0] = x;
    dragRect[2] = pressX - x;
  }
```

Continued on next page

The code *(continued)*

```
    if (y > pressY) {
      dragRect[1] = pressY;
      dragRect[3] = y - pressY;
    }
    else {
      dragRect[1] = y;
      dragRect[3] = pressY - y;
    }

   // Draw the new rectangle.
    panelGraphics.drawRect(dragRect[0], dragRect[1],
      dragRect[2], dragRect[3]);
  }

  /**
   * Zoom in or out.
   *
   * Called when releasing a mouse button.
   */
  public void zoom() {
    if (dsp == null) {
      return;
    }

    if (panelGraphics == null) {
      return;
    }

   // When the right mouse button was released, zoom
   // out with a factor 2.
    if (Uiobj.cO.state.getInt("mb") == 4) {
      dsp.zoomout();
    }
```

Continued on next page

Write a layer viewer, ***Continued***

The code *(continued)*

```
// Zoom in.
 else {
  // Remove the drag rectangle.
   if (dragRect != null) {
     panelGraphics.drawRect(dragRect[0], dragRect[1],
       dragRect[2], dragRect[3]);
   }
   dragRect = null;

  // Get the zoom coordinates.
   int x = Uiobj.cO.state.getInt("x");
   int y = Uiobj.cO.state.getInt("y");
   int pressX = Uiobj.cO.state.getInt("pressx");
   int pressY = Uiobj.cO.state.getInt("pressy");

  // The mouse was not moved, zoom with factor 2.
   if ((x == pressX) && (y == pressY)) {
     int width = dwa.width();
     int height = dwa.height();
     dsp.zoomin(x - width/4, y - height/4,
                x + width/4, y + height/4);
   }
   else {
     dsp.zoomin(pressX, pressY, x, y);
   }
 }

 // Redraw the layer with the new settings.
  changeLayer();
}
```

Continued on next page

The code *(continued)*

```
/**
 * Restore the total view of the layer.
 *
 * Called when the totalview button was clicked.
 */
public void totalView() {
  if (dsp == null) {
    return;
  }

  layerName = null;
  changeLayer();
}

/**
 * Change the skeleton setting.
 *
 * Called when the skeleton button was clicked.
 */
public void skeleton() {
  if (dsp == null) {
    return;
  }

  if (skeletonButton.isSelected()) {
    dsp.setoptions("skeleton");
  }
  else {
    dsp.setoptions("solid");
  }

  changeLayer();
}
```

Continued on next page

Write a layer viewer, ***Continued***

The code *(continued)*

```
  /**
   * Refreshes and shows the window.
   *
   * Called when the menu item is selected.
   */
  public static void showDialog() {
    if (dialog == null) {
      dialog = new LayerViewer();
    }

    dialog.refresh();
    dialog.show();
  }
}
```

Store the file as ***sources/com/company/ucam/layer/LayerViewer.java*** in the HOME directory.

```
/*
 * UcamActions.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the LayerViewer class.
import com.company.ucam.layer.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;
```

Continued on next page

Write a layer viewer, ***Continued***

The code *(continued)*

```
/**
 * Class loaded by Ucam.
 * Only its static block is executed.
 */
public class UcamActions {

  // Static block executed when the class is loaded.
   static {

    // Action to associate with the menu item.
    // Its action() method is executed when the button is clicked.
    // Calls the showDialog() method from the
    // com.company.ucam.layer.LayerViewer class.
     Ucamaction act = new Ucamaction() {
       public void action() {
         LayerViewer.showDialog();
       }
     };

    // Adds a button called 'layer_viewer' with label
    // 'View Layers' to the hypertool menu.
     Ucamapp.cO.add_pb("layer_viewer",
       "View Layers", null, act,
       "hypertool_menu", false);
   }
}
```

Store the file as ***sources/UcamActions.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac -d $HOME -classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/layer/LayerViewer.java
sources/UcamActions.java
```

Continued on next page

Write a layer viewer, ***Continued***

Compile the code *(continued)*

- For Windows:

```
javac –d %HOME% –classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\layer\LayerViewer.java
sources\UcamActions.java
```

The *classes com/company/ucam/layer/LayerViewer.class* and *UcamActions.class* are now created in the HOME directory.

Run the application

Start UCAM.

A *View Layers* item was added to the *Hypertool* menu.

Open a job.

Select the *View Layers* item.

A new window pops up with the desired functionality.

Write a clearance measure utility

Task

Write an application that allows measuring the clearance between two objects in the UCAM main window.

The functionality should be callable from a menu item in the HyperTool menu.

The code

```
/*
 * ClearanceMeasure.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.clearance;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

// Additional User Interface packages needed.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * Implements a JDialog with the necessary buttons
 * and result field.
 */
public class ClearanceMeasure extends JDialog {

  // Only one dialog can be created. Once created, it is
  // recycled for later use.
   private static ClearanceMeasure dialog;
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
// The field to display the measure result.
private JTextField textField;

// The last dragged line.
private int[] dragLine;

// The Graphics to draw directly to the screen.
private Graphics graphics;

// The javaPeer of the main drawing area.
private JPanel panel;

// The DisplayPar of plane 1 of the drawing area.
private Displaypar dsp;

/**
 * Constructs a new dialog window and initializes all
 * values.
 */
public ClearanceMeasure() {

  // Creates and initializes the JDialog with the
  // Ucam main window as parent and 'Measure Clearance'
  // as title.
   super(Uiobj.getFrame(), "Measure Clearance");
   getContentPane().setLayout(new BorderLayout());
   ((JPanel)getContentPane()).setBorder(new EmptyBorder(5,
     5, 5, 5));

   JPanel resultPanel = new JPanel(new BorderLayout(5, 5));
   resultPanel.add(new JLabel("clearance"),
     BorderLayout.WEST);

   textField = new JTextField();
   textField.setEditable(false);
   resultPanel.add(textField, BorderLayout.CENTER);
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
    getContentPane().add(resultPanel, BorderLayout.CENTER);

    // Create a panel to display the control buttons.
    JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 5,
      5));
    buttonPanel.setBorder(new EmptyBorder(5, 0, 0, 0));

    // The measure button.
    JButton button = new JButton("Measure");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        setup();
      }
    });
    buttonPanel.add(button);

    // The cancel button.
    button = new JButton("Cancel");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        dialog.hide();
      }
    });
    buttonPanel.add(button);

    getContentPane().add(buttonPanel, BorderLayout.SOUTH);

    pack();
  }
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
/**
 * Prepares the main window to measure distances and display
 * rubber lines.
 *
 * Called when the measure button is clicked.
 */
public void setup() {

  // Action to perform when the user drags with a mouse
  // button down over the main window.
   Ucamaction dragAction = new Ucamaction() {
     public void action(double x1, double y1, double x2,
       double y2) {
       dragLine(x1, y1, x2, y2);
     }
   };

  // Action to perform when the user releases a mouse
  // button over the main window.
   Ucamaction releaseAction = new Ucamaction() {
     public void action(double x1, double y1, double x2,
       double y2) {
       showClearance(x1, y1, x2, y2);
     }
   };

  // Sets the current rubberband action.
  // Remains valid until another action is set.
   Ucamapp.cO.rubberband("Measure clearance", releaseAction,
     dragAction, true, true);
}
```

Continued on next page

The code *(continued)*

```
/**
 * Implements rubberbanding functionality.
 *
 * Called when moving the mouse over the drawing area with
 * a button held down.
 */
public void dragLine(double x1, double y1, double x2,
  double y2) {

 // Determine the graphics of the main drawing area.
  if (graphics == null) {
    Udwaplane dwa = Ucamapp.cO.getMainDrawingarea();
    panel = (JPanel)dwa.getJavaPeer();
    graphics = panel.getGraphics();
    graphics.setXORMode(Color.white);
  }

 // Erase the old line when there was one.
 // Erasing is done by drawing the line a second
 // time. The XOR setting in panelGraphics makes
 // sure this erases the previous line.
  if (dragLine != null) {
    graphics.drawLine(dragLine[0], dragLine[1],
      dragLine[2], dragLine[3]);
  }

  dragLine = new int[4];

  Displaypar dsp = Ucamapp.cO.curdsp();
  Upoint p = new Upoint(x1, y1);
  p = p.stow(dsp);
  p = Ucamapp.cO.snappoint(p);
  p = p.wtos(dsp);
  dragLine[0] = (int)p.x();
  dragLine[1] = (int)p.y();
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
    p = new Upoint(x2, y2);
    p = p.stow(dsp);
    p = Ucamapp.cO.snappoint(p);
    p = p.wtos(dsp);
    dragLine[2] = (int)p.x();
    dragLine[3] = (int)p.y();

   // Draw the new line.
    graphics.drawLine(dragLine[0], dragLine[1], dragLine[2],
      dragLine[3]);
  }

 /**
  * Calculates the clearance and displays it.
  *
  * Called when releasing a mouse button.
  */
 public void showClearance(double x1, double y1, double x2,
   double y2) {

   // Clear the drag line when there was one.
    if (dragLine != null) {
      graphics.drawLine(dragLine[0], dragLine[1],
        dragLine[2], dragLine[3]);
    }

    dragLine = null;

    if (Ucamv6.ucam_job == null) {
      Ucamapp.cO.warning("No current job.");
      return;
    }
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
  // Determine the two objects selected.
   Ulayer lay = Ucamv6.ucam_job.lay_in_plane(1);
   if (lay == null) {
     return;
   }
   Upoint p1 = new Upoint(x1, y1);
   Upoint p2 = new Upoint(x2, y2);
   Displaypar dsp = Ucamapp.cO.curdsp();
   double radius = dsp.stow_val(5);

   Uapeobj apeobj1 = lay.closestobj(p1, radius);
   Uapeobj apeobj2 = lay.closestobj(p2, radius);

  // Calculate the clearance when 2 objects were selected.
   if ((apeobj1 != null) && (apeobj2 != null)) {
     textField.setText(DTLBuiltin.string(
       apeobj1.clearance_to(apeobj2)));
   }
   else {
     textField.setText("");
     Ucamapp.cO.warning("No object found.");
   }
   graphics = null;
 }

/**
 * Refreshes and shows the dialog.
 *
 * Called when the menu item is selected.
 */
public static void showDialog() {
   if (dialog == null) {
     dialog = new ClearanceMeasure();
   }
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
dialog.textField.setText("");
    dialog.show();
  }
}
```

Store the file as ***sources/com/company/ucam/clearance/ClearanceMeasure.java*** in the HOME directory.

```
/*
 * UcamActions.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the ClearanceMeasure class.
import com.company.ucam.clearance.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Class loaded by Ucam.
 * Only its static block is executed.
 */
public class UcamActions {

  // Static block executed when the class is loaded.
  static {

    // Action to associate with the menu item.
    // Its action() method is executed when the button
    // is clicked.
    // Calls the showDialog() method from the
    // com.company.ucam.clearance.ClearanceMeasure class.
```

Continued on next page

Write a clearance measure utility, ***Continued***

The code *(continued)*

```
    Ucamaction act = new Ucamaction() {
      public void action() {
        ClearanceMeasure.showDialog();
      }
    };

   // Adds a button called 'measure_clearance' with label
   // 'Measure Clearance' to the hypertool menu.
    Ucamapp.cO.add_pb("measure_clearance",
      "Measure Clearance", null, act,
      "hypertool_menu", false);
  }
}
```

Store the file as ***sources/UcamActions.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac –d $HOME –classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/clearance/ClearanceMeasure.java
sources/UcamActions.java
```

- For Windows:

```
javac –d %HOME% –classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\clearance\ClearanceMeasure.java
sources\UcamActions.java
```

The *classes com/company/ucam/clearance/ClearanceMeasure.class* and *UcamActions.class* are now created in the HOME directory.

Run the application

Start UCAM.

A *Measure Clearance* item was added to the *Hypertool* menu.

Open a job.

Select the *Measure Clearance* item.

A new window pops up.

Continued on next page

Run the application *(continued)*	Click the *Measure* button. Drag the mouse over the main window. A rubberline appears. Release the mouse button. The calculated measure is displayed.

Chapter 5 – Custom Panels

Overview

Introduction

Ucam already has extensive support for the creation of panels through the PanelPlus product, but it is possible to either modify the functionality of the PanelPlus product, or add completely new panel modules.

Contents

This chapter contains the following topics.

Creating a Panel module

Description

UCAM allows to add a menu item in the Panel menu cascade through the *upanel.subfiles* entry in *ucam.db* and the *UcamPanels* class.

upanel.subfiles

The *ucam.db* entry *upanel.subfiles* is a comma-separated list of custom panel modules to be added to the Panel cascade after the *PanelPlus* menu item.

For each entry, a menu item is added with the same name as the entry. White space before and after the entry is ignored, but the entry itself is case sensitive.

The action to be performed when selecting the item is defined in the *UcamPanels* class.

The UcamPanels class

For each of the items added through the *upanel.subfiles* entry, the *UcamPanels* class should have a class method defined with the exact same name appended with '_init'. This method is executed when the corresponding item is selected.

UCAM looks for the class in the following order:

- The user HOME directory
- The ETSCAM_CFG directory
- The ETSCAM_DAT directory
- The classes directory in the UCAM installation directory

Write a Step and Repeat module

Task

Write an application which implements a simple step and repeat given the repeat and clearance values.

The code

```
/*
 * StepRepeat.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.panels;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

// Additional User Interface packages needed.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * Implements a JDialog with the necessary buttons
 * and input fields.
 */
public class StepRepeat extends JDialog {

  // Only one dialog can be created. Once created, it is
  // recycled for later use.
  private static StepRepeat dialog = null;
```

Continued on next page

The code *(continued)*

```
// The input fields for repeat and clearance values.
private JTextField repeatX;
private JTextField repeatY;
private JTextField clearanceX;
private JTextField clearanceY;

/**
 * Constructs a new dialog window and initializes all values.
 */
public StepRepeat() {

  // Creates and initializes the JDialog with the
  // Ucam main window as parent and 'Step-Repeat' as title.
   super(Uiobj.getFrame(), "Step-Repeat");

   getContentPane().setLayout(new BorderLayout(5, 5));
   ((JPanel)getContentPane()).setBorder(new EmptyBorder(5,
     5, 5, 5));

  // Create a panel for the input fields and their labels.
   JPanel topPanel = new JPanel(new BorderLayout(5, 5));
   JPanel labelPanel = new JPanel(new GridLayout(2, 1, 5,
     5));
   labelPanel.add(new JLabel("Repeat"));
   labelPanel.add(new JLabel("Clearance"));
   topPanel.add(labelPanel, BorderLayout.WEST);

   JPanel textFieldPanel = new JPanel(new GridLayout(2, 2,
     5, 5));
   repeatX = new JTextField();
   textFieldPanel.add(repeatX);
   repeatY = new JTextField();
   textFieldPanel.add(repeatY);
   clearanceX = new JTextField();
   textFieldPanel.add(clearanceX);
```

Continued on next page

The code *(continued)*

```
clearanceY = new JTextField();
textFieldPanel.add(clearanceY);
topPanel.add(textFieldPanel, BorderLayout.CENTER);

getContentPane().add(topPanel, BorderLayout.NORTH);

// Create a panel for the control buttons.
JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 5,
  5));

// The OK button.
JButton button = new JButton("OK");
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    dialog.apply();
    dialog.hide();
  }
});
buttonPanel.add(button);

// The apply button.
button = new JButton("Apply");
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    dialog.apply();
  }
});
buttonPanel.add(button);
```

Continued on next page

Write a Step and Repeat module, ***Continued***

The code *(continued)*

```
    // The cancel button.
    button = new JButton("Cancel");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        dialog.hide();
      }
    });
    buttonPanel.add(button);

    getContentPane().add(buttonPanel, BorderLayout.SOUTH);

    pack();
  }

  /**
   * Perform a step and repeat based upon the input values.
   *
   * Called when the OK or Apply button is clicked.
   */
  public void apply() {
    int repX;
    int repY;
    double clearX;
    double clearY;
    int apeNum;
    int numLayers;
    Ulayer lay;
    Ubloape block;
    Urectangle jobSize;
    double stepX;
    double stepY;
    Ulayer blockLayer;
```

Continued on next page

Write a Step and Repeat module, ***Continued***

The code *(continued)*

```
// Read in the parameters.
 try {
   repX = new Integer(repeatX.getText()).intValue();
   repY = new Integer(repeatY.getText()).intValue();
   clearX = new Double(
     clearanceX.getText()).doubleValue();
   clearY = new Double(
     clearanceY.getText()).doubleValue();
 } catch (NumberFormatException e) {
   Ucamapp.cO.warning("Invalid step-repeat parameters");
   refresh();
   return;
 }

 if (Ucamv6.ucam_job == null) {
   Ucamapp.cO.warning("No current job");
   return;
 }

// Calculate the step.
 jobSize = Ucamv6.ucam_job.enclosingbox("all");
 stepX = jobSize.xsize() + clearX;
 stepY = jobSize.ysize() + clearY;

 Ucamapp.cO.muri_setup(Ucamv6.ucam_job, "Step Repeat");

// For each of the layers, create a new block aperture
// containing all the information from the layer and flash it
// according to the step and repeat parameters.
 apeNum = Ucamv6.ucam_job.ape_max_number() + 1;
 numLayers = Ucamv6.ucam_job.numlayers();
 for (int i = 1; i <= numLayers; ++i) {
   lay = (Ulayer)Ucamv6.ucam_job.getlayer("all", null, i);
   if (!lay.active()) {
     continue;
   }
```

Continued on next page

Write a Step and Repeat module, ***Continued***

The code *(continued)*

```
      blockLayer = Ulayer.cO.create((String)lay.CLASS());
      blockLayer.copy(lay, "all");
      lay.erase("all");
      lay.ape_clean();
      block = (Ubloape)Ubloape.cO.create(apeNum, blockLayer);
      lay.addape(block);
      block.repeat(repX, stepX, 0, repY, stepY, 0);
    }

   // Repaint the main drawing area to display the step and repeat.
    Ucamapp.cO.total_view();
  }

 /**
  * Initializes the input fields.
  */
  public void refresh() {
    repeatX.setText("1");
    repeatY.setText("1");
    clearanceX.setText("0");
    clearanceY.setText("0");
  }

 /**
  * Refreshes and shows the dialog.
  *
  * Called when the menu item is selected.
  */
  public static void showDialog() {
    if (dialog == null) {
      dialog = new StepRepeat();
    }
```

Continued on next page

Write a Step and Repeat module, ***Continued***

The code *(continued)*

```
    dialog.refresh();
    dialog.show();
  }
}
```

Store the file as ***sources/com/company/ucam/panels/StepRepeat.java*** in the HOME directory.

```
/*
 * UcamPanels.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the custom panel classes.
import com.company.ucam.panels.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Class where Ucam looks for panel initialization methods.
 */
public class UcamPanels {

  // Method called for the StepRepeat module.
  public static void StepRepeat_init() {
    StepRepeat.showDialog();
  }
}
```

Store the file as ***sources/UcamPanels.java*** in the HOME directory.

Modify the upanel.subfiles entry

In *ucam.db* set the *upanel.subfiles* entry to *StepRepeat*.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac –d $HOME –classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/panels/StepRepeat.java
sources/UcamPanels.java
```

- For Windows:

```
javac –d %HOME% –classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\panels\StepRepeat.java
sources\UcamPanels.java
```

The *classes com/company/ucam/panels/StepRepeat.class* and *UcamPanels.class* are now created in the HOME directory.

Run the application

Start UCAM.

Open a job.

A *StepRepeat* item was added to the *Panel* cascade.

Select the *StepRepeat* item.

A new window pops up.

Enter the desired values.

Click the OK button. The StepRepeat window disappears.

Click the TotalView button to see the result of the StepRepeat operation.

Extending PanelPlus

Introduction

When *PanelPlus* supplies almost the correct functionality, it might be more interesting to modify PanelPlus instead of writing a complete new Panel module.

The necessary hooks are in place to customize the standard PanelPlus module and insert it into Ucam.

The Upanel class

The *Upanel* class represents the PanelPlus window with most of its functionality. By subclassing the *Upanel* class, overloading some of its methods and instantiating an instance of the new class, it is possible to customize its functionality to match specific needs. When overloading a method, one can call 'super()' to obtain the original functionality.

Upanel processing flow

When running a PanelPlus session a determined flow is followed.

When changing from the setups to the results panel, the following Upanel instance methods are called:

- *fill_job_list()*
- *calc_result(Upanelparam, Uframe, boolean, boolean)* : called for each of the frame jobs in the frame set.

When a result is selected in the results list, the following Upanel instance methods are called:

- *calc_offset(Uresult, Uframe, String)*
- *fill_coupon_list(Uframe, Uresult)*
- *calc_coupon_positions(Uframe, Uresult)*
- *make_background(Uresult, Uframe, double)*

When pressing DO, the following Upanel instance methods are called:

- *make_background(Uresult, Uframe, double)*
- For each of the active layers in the job:
  - *make_border(Uresult, Uframe, Ulayer, Ulayer, Dict)*
  - *add_before(Uresult, Uframe, Ulayer, Ulayer)*
  - *add_after(Uresult, Uframe, Ulayer, Ulayer)*
  - *add_coupons(Uframe, Ulayer, Ulayer)*
  - *change_layname(Ulayer)*
- *replace_text(Ujob, Dict, Uresult)*
- *sr_end(Uframe)*
- *change_jobname(Ujob)*

- public void fill_job_list()

Jobs to be panelized are filled in the job list represented by the ‘jobs’ instance variable of type Upanellist. The elements of the list are Upaneljob instances.

- Uresult cal_result(Upanelparam pan, Uframe frm, boolean mutli, boolean lshape)

Calculates the result for each frame in the frame list.

Parameters:

pan – defines the panel parameter to use
frm – defines the frame to use
multi – is true for multi jobs panels
lshape – is true when L-shape nesting is selected

The class Upanelparam carries the required variables defined during the input session .

(GUI of PanelPlus; prior switching to result)

It provides all necessary methods to get and set those variables. calc_result() returns a Uresult instance which contains information about a “Panelized” image.

(GUI of PanelPlus, result screen; each entry of the result list has its own Uresult)

It describes the step and repeat, clearance, rotation, total dimension, number of panels, fill and yield, offset, background layer and the group of flashpoints.

Called when result selected

- public void calc_offset(Uresult res, Uframe frm, String pos, String place)

Calculates and fills in the offset for Uresult. The offset is the absolute coordinate of the bottom left point of the total step and repeat image. It gets called when selecting a result in the result list.

Parameters:

res – should be one of the modified results obtained from calc_result()
frm – should be the same frame as used for result calculation
pos – “relative” or “datum”
place - ???

- public void fill_coupons_list(Uframe frm, Uresult res)

Fills the coupon list with all coupons defined in the coupon setup file. For each coupon a Ucoupon instance is added to the coupon list. The positions, place, kind, clearance and distance will be added to each coupon. The calculation of the absolute coordinates of the coupon is done during panelization. It gets called when selecting a result in the result list.

Parameters:

frm – The same frame to use.
res – The same result to use.

The following example shows how to parse the coupons list during panelize and modify as required.

```
nCoupons = coupons.count();
for (int I = 1; I <= nCoupons; ++I) {
  Ucoupon coupon = (Ucoupon)coupons.get(I);
  // one can call methods of the cureent coupon eg.
  // coupon.clearpos()
}
```

See DrillCouponPanel example on how to generate and add coupns.

- public void calc_coupons_positions(Uframe frm, Uresult res)

Calcuates the absolute flashpoints of the coupons based on the relative indications and the result information. It gets called when selecting a result in the result list.

Parameters:

frm – The used frame.
res – The used result.

- public void make_background(Uresult res, Ufrme frm, double routclr)

Calculates and sets the backlay of the currently used result. This method creates a layer with a contour aperture defining the background of the panelized image. The form defined in this layer will be cut out of the frame before adding the job blocks. It gets called when selecting a result in the result list to draw in the preview.

Parameters:

res – The used result
frm – The used frame
routclr – The rout clearance

When pressing DO in PanelPlus result GUI, the following Upanel instance methods are called:

public void make_background(Uresult res, Uframe frm, double routclr)

PanelPlus and its public hooks, ***Continued***

For each active layer in the job Ucam calls the following methods:

Called for each layer

- public void make_border(Uresult res, Uframe frm, Ulayer out_lay, Ulayer lay, Dict txtdb)

Creates and out_lay which contains the border that encloses the panelized image. Gets called for each layer.

Parameters:

res – The used result
frm – The used frame
out_lay – The layer to be used
lay – The job layer that is currently panelized
txtdb – A dictionary with 5 labels with information from the setup file The symbols are the aperture number specified in the setup file, the corresponding values are the strings entered in the text fields of the panel editor.

- public void add_before(Uresult res, Uframe frm, Ulayer out_lay, Ulayer inp_lay)

Dummy method, added to have another hook for customizing. Gives the opportunity to add special features to the border layer (out_lay) created in the make_border method. Gets called for each layer.

This method is called before the actual input of the job blocks.

Parameters:

res – The used result
frm – The used frame
out_lay – The layer to be used
inp_lay – The job layer that is currently panelized

- public void add_coupons(Uframe frm, Uayer outlay, Ulayer lay)

Adds all the coupons out of the coupons list to the outlay. The positions of the coupons were calculated in the calc_coupon_position method. If the backlayer contains information at this position a negative rectangle is cut out of the outlay. The size of the rectangle is the enclosing box of the coupon job (spread a little bit). If the coupon job contains an outline layer, the enclosing box of that layer is taken. Gets called for each layer.

Parameters:

frm – The used frame
out_lay – The layer contains frame + step&repeat image
lay – The layer; only the single image

PanelPlus and its public hooks, ***Continued***

- public String change_layname(Ulayer lay)

Sets the name of the layer after panelization. Gets called for each layer.

The following methods get called once after panlization:

Called once

- public void replace_text(Ujob job, Dict txtdb, Uresult res)

Replaces text apertures according to the codes set in the set file and the text labels defined in the dictionary txtdb. The symbols in this dictionary are the aperture numbers specified in the setup file. The corresponding values are the Strings entered in the text fill-ins of the panel editor. Gets called when the complete job is panelized.

Parameters:

job – The panelized job
txtdb – The dictionary with the strings
res – The current result

- public void sr_end(Uframe frm)

Dummy method, added to have another hook for customizing. Gives the possibility to make final changes to the job that is output. At this point, the panelized job is the ucam_job.

Parameter:

frm – The used frame

- public String change_jobname(Ujob job)

Sets the name of the job after panelization.

Parameter:

job – The current job

Exercise: PrintPanel

Task

Make a *Upanel* subclass, which retains the original PanelPlus functionality, but prints out a message during each step.

The code

```
/*
 * PrintPanel.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.panels;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Custom Upanel subclass which prints out a message
 * when each of the Upanel methods is called.
 */
public class PrintPanel extends Upanel$CO {
  public void fill_job_list() {
    System.out.println("fill_job_list()");
    super.fill_job_list();
  }

  public Uresult calc_result(Upanelparam pan, Uframe frm,
boolean multi, boolean lshape) {
    System.out.println("calc_result(Upanelparam, Uframe,
boolean, boolean)");
    return super.calc_result(pan, frm, multi, lshape);
  }
```

Continued on next page

Exercise: PrintPanel, ***Continued***

The code *(continued)*

```
  public void calc_offset(Uresult res, Uframe frm, String
pos, String place) {
    System.out.println("calc_offset(Uresult, Uframe,
String)");
    super.calc_offset(res, frm, pos, place);
  }

  public void calc_offset(Uresult res, Uframe frm, String
pos, Upoint place) {
    System.out.println("calc_offset(Uresult, Uframe,
Upoint)");
    super.calc_offset(res, frm, pos, place);
  }

  public void fill_coupon_list(Uframe frm, Uresult res) {
    System.out.println("fill_coupon_list(Uframe, Uresult)");
    super.fill_coupon_list(frm, res);
  }

  public void calc_coupon_positions(Uframe frm, Uresult res)
{
    System.out.println("calc_coupon_positions(Uframe,
Uresult)");
    super.calc_coupon_positions(frm, res);
  }

  public void make_background(Uresult res, Uframe frm, double
routclr) {
    System.out.println("make_background(Uresult, Uframe,
double)");
    super.make_background(res, frm, routclr);
  }
```

Continued on next page

Exercise: PrintPanel, *Continued*

The code *(continued)*

```
  public void make_border(Uresult res, Uframe frm, Ulayer
out_lay, Ulayer lay, Dict txtdb) {
    System.out.println("make_border(Uresult, Uframe, Ulayer,
Ulayer, Dict)");
    super.make_border(res, frm, out_lay, lay, txtdb);
  }

  public void add_before(Uresult res, Uframe frm, Ulayer
out_lay, Ulayer inp_lay) {
    System.out.println("add_before(Uresult, Uframe, Ulayer,
Ulayer)");
    super.add_before(res, frm, out_lay, inp_lay);
  }

  public void add_after(Uresult res, Uframe frm, Ulayer
out_lay, Ulayer inp_lay) {
    System.out.println("add_after(Uresult, Uframe, Ulayer,
Ulayer)");
    super.add_after(res, frm, out_lay, inp_lay);
  }

  public void add_coupons(Uframe fr, Ulayer outlay, Ulayer
lay) {
    System.out.println("add_coupons(Uframe, Ulayer,
Ulayer)");
    super.add_coupons(fr, outlay, lay);
  }

  public String change_layname(Ulayer lay) {
    System.out.println("change_layname(Ulayer)");
    return super.change_layname(lay);
  }
```

Continued on next page

Exercise: PrintPanel, ***Continued***

The code *(continued)*

```
  public void replace_text(Ujob job, Dict txtdb, Uresult res)
{
    System.out.println("replace_text(Ujob, Dict, Uresult)");
    super.replace_text(job, txtdb, res);
  }

  public void sr_end(Uframe frm) {
    System.out.println("sr_end(Uframe)");
    super.sr_end(frm);
  }

  public String change_jobname(Ujob job) {
    System.out.println("change_jobname(Ujob)");
    return super.change_jobname(job);
  }
}
```

Store the file as ***sources/com/company/ucam/panels/PrintPanel*** in the HOME directory.

```
/*
 * UcamPanels.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the custom panel classes.
import com.company.ucam.panels.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;
```

Continued on next page

Exercise: PrintPanel, ***Continued***

The code *(continued)*

```
/**
 * Class where Ucam looks for panel initialization methods.
 */
public class UcamPanels {

  // Method called for the PrintPanel module.
  public static void PrintPanel_init() {
    Ucamv6.u$panel = new PrintPanel();
    ((Upanel.CO)Ucamv6.u$panel).run("PrintPanel");
  }
}
```

Store the file as ***sources/UcamPanels.java*** in the HOME directory.

Modify the upanel.subfiles entry

In *ucam.db* set the *upanel.subfiles* entry to *PrintPanel*.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac -d $HOME -classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/panels/PrintPanel.java
sources/UcamPanels.java
```

- For Windows:

```
javac -d %HOME% -classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\panels\PrintPanel.java
sources\UcamPanels.java
```

The *classes com/company/ucam/panels/PrintPanel.class* and *UcamPanels.class* are now created in the HOME directory.

Run the application

Start UCAM.

Open the *cad* job.

Create a Frame set with the *panel1* and *panel2* jobs.

A *PrintPanel* item was added to the *Panel* cascade.

Continued on next page

Exercise: PrintPanel, ***Continued***

Run the application *(continued)*

Select the *PrintPanel* item.

Create a panel using the Frame set.

When running through the PanelPlus flow, the messages are print out in the terminal window.

Exercise: DrillCouponPanel

Task

Make a *Upanel* subclass, which defines a coupon called “DrillCoupon” containing a single flash of each of the used circular apertures of the drill layers.

The code

```
/*
 * DrillCouponPanel.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.panels;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Custom Upanel subclass which overloads the
 * fill_coupon_list method.
 */

public class DrillCouponPanel extends Upanel$CO {
  /**
   * Fills up the coupons list with all the coupons
   * for this panel.
   * A new Ucoupon is constructed for the 'DrillCoupon'
   * coupon.
   */
```

Continued on next page

Exercise: DrillCouponPanel, ***Continued***

The code *(continued)*

```
public void fill_coupon_list(Uframe frm, Uresult res) {
  super.fill_coupon_list(frm, res);

  if (coupons == null) {
    return;
  }

  Ucoupon coupon = new Ucoupon("DrillCoupon");
  coupon.setjob(make_coupon_job(frm));
  coupons.add(coupon);
}

/**
 * Create a job for the DrillCoupon.
 * The job contains one drillLayer.
 */
public Ujob make_coupon_job(Uframe frm) {

  // Create a drill layer.
  Udrilayer couponLayer = (Udrilayer)Udrilayer.cO.create();
  if (couponLayer == null) {
    return null;
  }

  // Create a job.
  Ujob couponJob = Ujob.cO.create();
  if (couponJob == null) {
    return null;
  }

  couponLayer.setactive(true);

  // For each of the drill layers in the job to panelize,
  // add the drill apertures to the couponLayer.
  Ujob job;
  Udrilayer lay;
```

Continued on next page

Exercise: DrillCouponPanel, ***Continued***

The code *(continued)*

```
    for (int i = 1; i <= jobs.count(); ++i) {
      job = (Ujob)jobs.get(i);
      for (int j = 1; j <= job.numlayers("drill"); ++j) {
        lay = (Udrilayer)job.getlayer("drill", "", j);
        add_drills(lay, couponLayer);
      }
    }

   // Flash each of the drill apertures once.
    Uape ape = couponLayer.firstape();
    double pos = 0;

    for (int i = 1; i <= couponLayer.numapes(); ++i) {
      ape.flash(0, pos);
      pos += 100*Ucamv6._PCT_mil();
      ape = ape.next();
    }

   // Add the resulting layer to the job.
    couponJob.addlayer(couponLayer);
    return couponJob;
  }

 /**
  * Look for all the flashed circular apertures in the layer
  * and add them to the couponLayer.
  */
 public void add_drills(Udrilayer lay, Udrilayer
    couponLayer) {
    Uape ape = lay.firstape();
    Uape cirApe;
```

Continued on next page

Exercise: DrillCouponPanel, ***Continued***

The code *(continued)*

```
    for (int i = 1; i <= lay.numapes(); ++i) {
      if (ape.is_it("cir") && !ape.reverse() &&
      (ape.numobj("f") != 0)) {
        cirApe = couponLayer.apesearch(ape, "");
        if (cirApe == null) {
          cirApe = ape.copydef();
          couponLayer.addape(cirApe);
        }
      }
      ape = ape.next();
    }
  }
}
```

Store the file as ***sources/com/company/ucam/panels/DrillCouponPanel*** in the HOME directory.

```
/*
 * UcamPanels.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Allows to locate the custom panel classes.
import com.company.ucam.panels.*;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

/**
 * Class where Ucam looks for panel initialization methods.
 */
```

Continued on next page

Exercise: DrillCouponPanel, ***Continued***

The code *(continued)*

```
public class UcamPanels {

  // Method called for the DrillCouponPanel module.
  public static void DrillCouponPanel_init() {
    Ucamv6.u$panel = new DrillCouponPanel();
    ((Upanel.CO)Ucamv6.u$panel).run("DrillCouponPanel");
  }
}
```

Store the file as ***sources/UcamPanels*** in the HOME directory.

Modify the upanel.subfiles entry

In *ucam.db* set the *upanel.subfiles* entry to *DrillCouponPanel*.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac –d $HOME –classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/panels/DrillCouponPanel.java
sources/UcamPanels.java
```

- For Windows:

```
javac –d %HOME% –classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\panels\DrillCouponPanel.java
sources\UcamPanels.java
```

The *classes com/company/ucam/panels/DrillCouponPanel.class* and *UcamPanels.class* are now created in the HOME directory.

Run the application

Start UCAM.

Open the *cad* job.

Create a Frame set with the *panel1* and *panel2* jobs.

Create an empty Coupon set.

A *DrillCouponPanel* item was added to the *Panel* cascade.

Continued on next page

Exercise: DrillCouponPanel, ***Continued***

Run the application *(continued)*

Select the *DrillCouponPanel* item.

Create a panel using the Frame set.

In the lower left corner of the resulting panelized job, a row of circular flashes has been added.

Chapter 6 – Standalone UCAM modules

Overview

Introduction

UCAM has great functionality accessible from within the UCAM user interface. In some cases it is desirable to use the UCAM functionality without the need for the interface. This chapter shows how to do this.

NOTE: This functionality will only be available from UCAM v6.1-3 on.

Contents

This chapter contains the following topics.

The UCAM setup method

Description

When starting UCAM, a number of steps are performed to initialize and load the necessary libraries (both Java and native). This setup step can be called using the ***Ucam.setup()*** method. This is a class method, which can be called upon the *Ucam* class.

Exercise: JobInfo

Task

Write an application, which allows the user to open a job file and displays the information in a window.

The code

```
/*
 * JobInfo.java
 *
 * Copyright (c) 2016 Ucamco NV All rights reserved.
 */

// Defines the package this class resides in.
// This should be the first line of code in the source file.
package com.company.ucam.jobinfo;

// Standard Ucam packages.
import com.barco.ets.ucam.dtl.*;
import com.barco.ets.ucam.ui.*;
import com.barco.ets.ucam.hypertool.*;

// Need to import because in the standard package.
import Ucam;

// Additional packages needed.
import javax.swing.*;
import java.awt.event.*;
import java.io.*;

/**
 * Implements a window with a menubar and an area to display
 * the job information.
 */
public class JobInfo extends JFrame {

  // The info area.
   JTextArea textArea;
```

Continued on next page

Exercise: JobInfo, ***Continued***

The code *(continued)*

```
// The fileChooser for the open and save buttons.
JFileChooser fileChooser;

/**
 * Starts up the application.
 */
public static void main(String[] args) {
  Ucam.setup();

 // Create a new window and display it.
  new JobInfo().setVisible(true);
}

/**
 * Constructs a new frame and creates the user interface.
 */
public JobInfo() {

 // Create a window with title 'JobInfo'.
  super("JobInfo");

 // Make sure the application stops when the user closes
 // the window.
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }
  });

 // Create a menubar with a 'File' menu.
 // The file menu has 3 items :
 // - Open
 // - Save
 // - Quit
```

Continued on next page

The code *(continued)*

```
JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");

JMenuItem item = new JMenuItem("Open");
item.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    openJob();
  }
});
fileMenu.add(item);

item = new JMenuItem("Save");
item.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    saveText();
  }
});
fileMenu.add(item);

fileMenu.addSeparator();

item = new JMenuItem("Quit");
item.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.exit(0);
  }
});
fileMenu.add(item);

menuBar.add(fileMenu);

setJMenuBar(menuBar);
```

Continued on next page

Exercise: JobInfo, ***Continued***

The code *(continued)*

```
    // Create the info area.
    textArea = new JTextArea(15, 80);
    textArea.setEditable(false);

    // Add the info area through a scrollpane to the window.
    getContentPane().add(new JScrollPane(textArea));

    // Initialize the filechooser.
    fileChooser = new JFileChooser();

    pack();
  }

  /**
   * Asks the user to select a job, opens it and displays the
   * information.
   *
   * Called when the 'Open' item is selected.
   */
  private void openJob() {

    // Check if a file was selected.
    if (fileChooser.showOpenDialog(this) ==
    JFileChooser.APPROVE_OPTION) {
      String fileName = null;

      try {
        fileName = fileChooser.getSelectedFile()
          .getCanonicalPath();
      } catch (Exception e) {
        return;
      }
```

Continued on next page

Exercise: JobInfo, ***Continued***

The code *(continued)*

```
    // Read in the job.
     Ujob job = Ujob.cO.read(fileName);

     if (job == null) {
       System.out.println("Invalid job file");
       return;
     }

    // Get information regarding the job.
     textArea.setText("Job name : " + job.name() + "\n");
     textArea.append("Customer : " + job.customer() + "\n");

     int numLayers = job.numlayers();
     textArea.append("Number of layers : " + numLayers +
       "\n\n");

     Ucamobj layer;

    // Get information for each layer.
     for (int i = 1; i <= numLayers; ++i) {
       layer = job.getlayer("all", "", i);
       textArea.append("Layer " + i + " : " + layer.name() +
         "\n");
       textArea.append("File specification : " +
         layer.spec() + "\n");

       textArea.append("\n");
     }
   }
}
```

Continued on next page

Exercise: JobInfo, *Continued*

The code *(continued)*

```
  /**
   * Asks the user for a file to save the job information to.
   *
   * Called when the 'Save' item is selected.
   */
  private void saveText() {
    if (fileChooser.showSaveDialog(this) ==
    JFileChooser.APPROVE_OPTION) {
      File file = fileChooser.getSelectedFile();
      file.getParentFile().mkdirs();

      try {
        PrintWriter pw = new PrintWriter(new
          BufferedWriter(new FileWriter(file)));
        pw.print(textArea.getText());
        pw.close();
      } catch (IOException e) {
      }
    }
  }
}
```

Store the file as ***com/company/ucam/jobinfo/JobInfo.java*** in the HOME directory.

Compile the code

From the HOME directory, run the Java compiler.

- For Unix:

```
javac -d $HOME -classpath
$HOME:$ETSCAM_INSTALL/ucam/classes/ucam.jar
sources/com/company/ucam/jobinfo/JobInfo.java
```

- For Windows:

```
javac -d %HOME% -classpath
%HOME%;%ETSCAM_INSTALL%\ucam\classes\ucam.jar
sources\com\company\ucam\jobinfo\JobInfo.java
```

The class *com/company/ucam/jobinfo/JobInfol.class* is now created in the HOME directory.

Run the application

To start up the application a number of environment variables need to be set up correctly. This is done in the Ucam start up script (*ucam* for Unix platforms, *ucam.bat* for Windows systems).

Make a copy of the ucam start up script. In the line where *java* is called, replace *Ucam* with *com.company.ucam.jobinfo.JobInfo*.

Run the new script.

A window appears.

Select the *Open* item from the *File* menu.

Select a .job file, information regarding the selected job appears in the window.

Select the *Save* item from the *File* menu.

Select a file to store the information in.
