The Eclipse Scout Book

Version 3.9 (Kepler)

Published by BSI Business Systems Integration AG

The Eclipse Scout Book

Release 3.9 (Kepler)

Matthias Zimmermann

Version of 2014-02-26

Contents

Preface v

I Getting Started 1

1 Introduction 3

1.1 What is Scout? . . . 3

1.1.1 End User Perspective . . . 3

1.1.2 Management Perspective . . . 6

1.1.3 Developer Perspective . . . 8

1.2 Why should I choose Scout? . . . 11

1.2.1 Get up to speed in days . . . 11

1.2.2 Run your apps on web, desktop and mobile . . . 11

1.2.3 Scout is production ready . . . 11

1.2.4 Protect your investment . . . 11

1.3 What should I read? . . . 11

1.3.1 I know some Java . . . 12

1.3.2 I know tons of both Java and Eclipse . . . 13

1.3.3 I am a manager . . . 13

2 ”Hello World” Tutorial 15

2.1 Installation and Setup . . . 15

2.2 Create a new Project . . . 15

2.3 Run the Initial Application . . . 16

2.4 The User Interface Part . . . 19

2.5 The Server Part . . . 22

2.6 Add the Rayo Look and Feel . . . 23

2.7 Exporting the Application 25
2.8 Deploying to Tomcat 27

3 "Hello World" Background **31**
3.1 Create a new Project 31
3.2 Walking through the Initial Application 33
  3.2.1 Desktop 34
  3.2.2 Form 34
  3.2.3 Form Handler 35
  3.2.4 Form Services and Form Data Objects 37
3.3 Run the Initial Application 37
  3.3.1 The Launcher Boxes 37
  3.3.2 Eclipse Product Files 37
  3.3.3 Eclipse Configuration Files 39
  3.3.4 Scout Desktop Client Applications 41
  3.3.5 Scout Web, Tablet and Mobile Clients 41
3.4 The User Interface Part 42
3.5 The Server Part 44
  3.5.1 Scout Services 44
  3.5.2 Scout Proxy Services 45
3.6 Add the Rayo Look and Feel 47
3.7 Exporting the Application 47
3.8 Deploying to Tomcat 50

4 Scout Tooling **53**
4.1 The Eclipse IDE 53
4.2 The Scout SDK 54
4.3 The Scout Explorer 56
4.4 The Scout Object Properties 59
4.5 Scout SDK Wizards 62
  4.5.1 Creating a new Scout Project 62
  4.5.2 Exporting a Scout Project 64
  4.5.3 Creating new Form Fields 66
  4.5.4 The NLS Editor 69

5 A Larger Example 71
5.1 The ”My Contacts” Application 71
5.2 Setting up the new Scout project 74
5.3 Adding the Person Page 80
5.4 Adding the Company Page 84
5.5 Installing the Database 85
5.5.1 Setting up the Database Schema 86
5.5.2 Scout Logging 91
5.6 Fetching Data from the Database 91
5.7 Creating the Person Form 92
5.7.1 Creating the Form Layout with From Fields 94
5.7.2 A simple Form to edit the Picture URL 97
5.7.3 Linking the Person Form to the Person Page 99
5.7.4 Adding the Company Smartfield 99
5.7.5 Adding the Map Form 103
5.8 Managing Person Data on the Server Side 106
5.9 Creating the Company Form 109
5.10 Adding the Scribe Library to the Application 110
5.11 Integrating LinkedIn Access with Scribe 111
5.12 Fetching Contacts from LinkedIn 119

II Appendices 125

A Licence and Copyright 127
A.1 Licence Summary 127
A.2 Contributing Individuals 128
A.3 Full Licence Text 128

B Scout Installation 137
B.1 Overview 137
B.2 Download and Install a JDK 137
B.2.1 Windows 137
B.3 Download and Install Scout 139
B.3.1 Windows 139

B.4 Add Scout to your Eclipse Installation . . . 143
B.5 Verifying the Installation . . . 144

C Apache Tomcat Installation **147**
C.1 Windows Installation . . . 147
C.2 Directories and Files . . . 148
C.3 The Tomcat Manager Application . . . 149

Bibliography **151**

Index **152**

Preface

Who should read this Book?

- You want to build a multi-user mobile application.
- You have to build a Java based business application.
- You need to decide on the IT strategy of your organisation
- You would like to show off the great library you have built but have hardly time for the good looking UI that it deserves

If any of the above scenarios is relevant in your situation, you might want to consider Section 1.3 on what to read and in which order.

This Book is Open and Free

- Access is simple. To download the book you do not need to register or pay subscription fees. To cater for individual preferences the book is available in various formats. Eventually, there should also be a printed version. And translations.
- Reuse/rehash is simple. This book is released under the Creative Commons (CC-BY) license. See Appendix A for details.
- Contribution is simple. That's why the source files are all available on Github and we maintain a continuous integration (CI) infrastructure for building the book. The necessary information to help you to contribute is available on the Scout wiki.[1]

In short, you are allowed to freely use, share and adapt this book, as long as you respect the conditions of the license mentioned above.

---

[1] The Scout book `http://wiki.eclipse.org/Scout/Book/`

The Scout Community

The community around Scout is its most valuable asset. It is active, growing and friendly. This is why it deserves to be mentioned right here. The Scout forum[2] is the central place for the Scout community to share ideas, ask questions and get help. Other, less interactive community resources are the Scout wiki pages[3], the Scout homepage[4], the blog[5], and twitter[6].

The simplest way to become a community member yourself is to start with downloading Scout from the Eclipse download site[7], installing the software as described in Appendix B, and continue reading here.

The Current State

With the Eclipse Kepler release a first (and still rough) version of the Eclipse Scout book is available. In its current state, this book provides a reasonable introduction into the Eclipse Scout framework. To allow for external contributions from the Scout community, the book's setup has been designed to invite collaboration[8]

Over time – and with each new Scout release – this book is intened to be updated and extended to cover more aspects with every release.

Thanks!

Many ideas regarding various aspects of this book, such the book's setup, were copied from freely available resources on the web. Significant impact had "Pharo by Example"[9].

Last but not least, we thank BSI Business Systems Integration AG[10] for its contribution of Eclipse Scout to the open source community. For its dedication to open standards and the support to further evolve Scout and its growing community.

---

[2] Scout forum `http://www.eclipse.org/forums/eclipse.scout`

[3] Scout wiki `http://wiki.eclipse.org/scout`

[4] Scout homepage `http://www.eclipse.org/scout`

[5] Scout blog `http://www.bsiag.com/scout`

[6] Scout twitter `http://twitter.com/EclipseScout`

[7] Scout download `http://www.eclipse.org/downloads/`

[8] The books technical setup and contributing work flow is described in the Scout wiki: `http://wiki.eclipse.org/Scout/Book/`.

[9] "Pharo by Example" `http://pharobyexample.org/`

[10] BSI Business Systems Integration AG `http://www.bsiag.com/`

Part I

Getting Started

Chapter 1

Introduction

1.1 What is Scout?

Scout is a mature framework for building business applications. With its multi-frontend support, a single Scout applications may run as a desktop application, in a web browser or on a mobile phone with touch support. The clean separation of Scout's client model from the user interface technologies allows the developer to concentrate on a single code base. This approach has advantages for end users of Scout applications, Scout developers, and organisations implementing Scout applications. In the text below we will explain Scout from the perspective of these three roles.

1.1.1 End User Perspective

End users of enterprise applications typically care about friendly user interfaces (UI) and well designed functionality that support them in doing their job. Depending on the role and the current context of an end users, either mobile, web or desktop clients work best. In the text below, we use a real world enterprise application to highlight how Scout applications can provide user friendly UIs on the desktop, in the browser and on mobile phones.

If the user is working from a desk and frequently engaged in customer calls, good integration with locally installed software such as Lotus Notes or Microsoft Office products can significantly boost productivity. So far, such a scenario still requires a desktop client application. An example screenshot of a Scout desktop client application from a customer relationship management (CRM) solution is provided in Figure 1.1. The screenshot provides a good overview of the general layout of the application. On the left side a tree for simple navigation through the available entities, such as companies, persons and communications is provided. Once an entity is selected, a click on the magnifying glass opens a search form on the right hand side. In this search form a number of search fields is provided. After entering "eclipse" into the company search field – and clicking the search button at the bottom of the search form – the list of matching entries is presented to the user. With a right mouse click on the desired row, the corresponding company dialog can be opened for editing.

When a user needs to work from a computer that does not have a desktop client installed (or lacks the time or the necessary permission to install software) the natural choice is to work with a web application. As Scout offers multi-frontend capabilities out of the box, a web frontend can

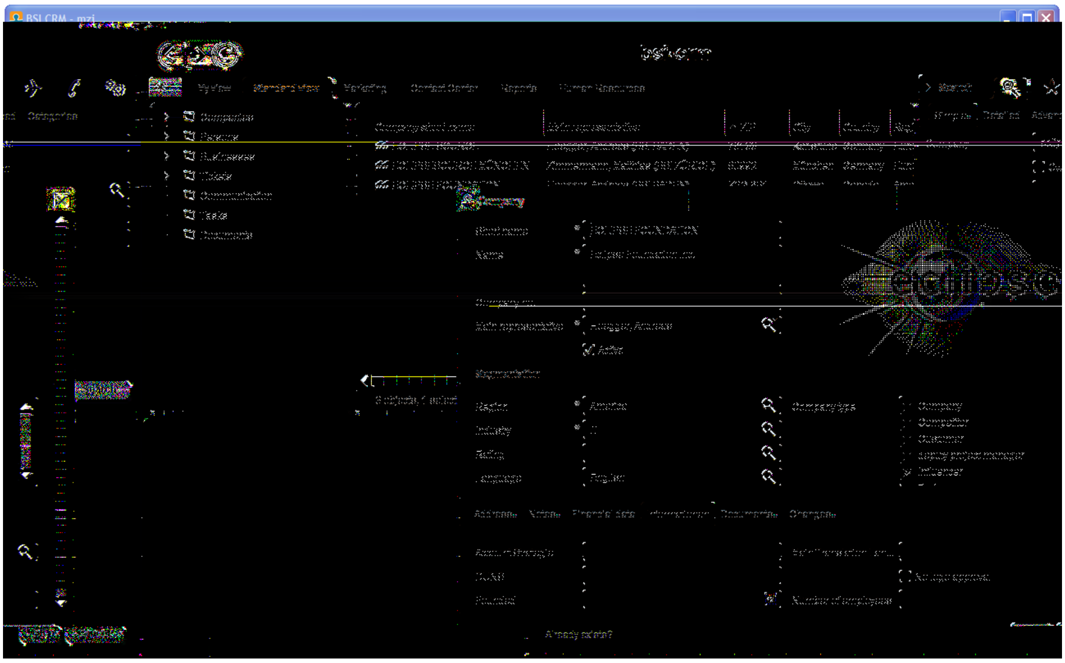
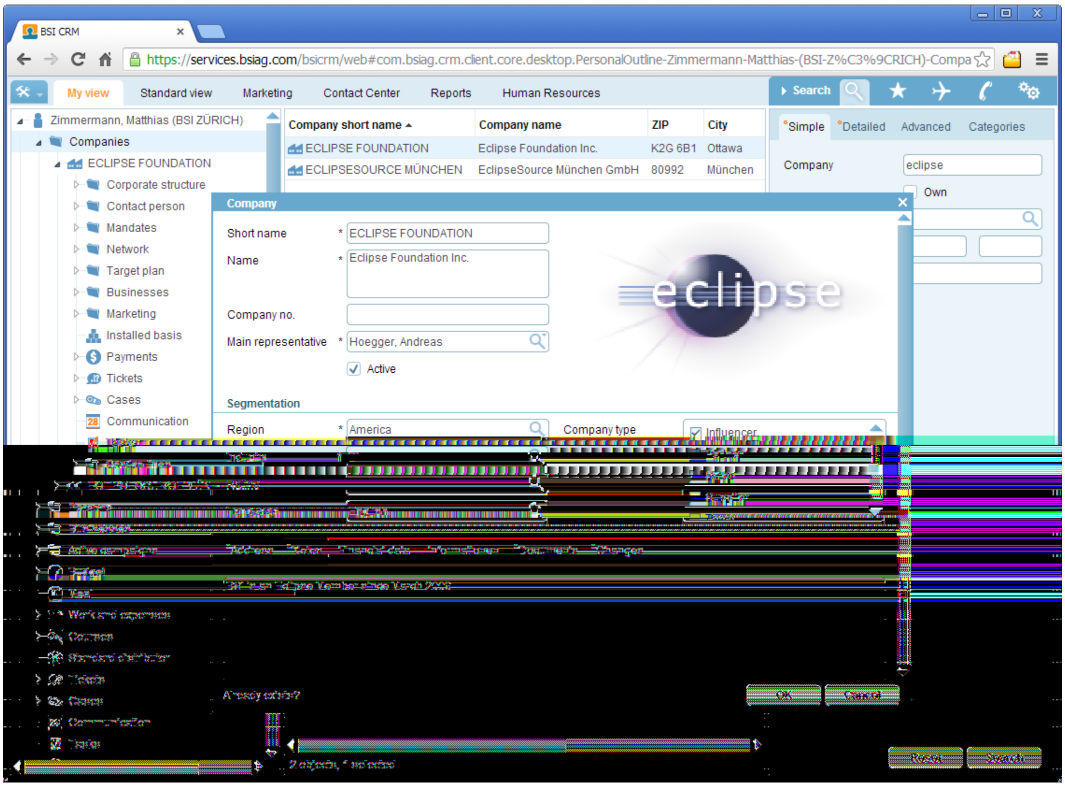


Figure 1.1: The desktop client of a Scout enterprise application.



Company short name	Company name	ZIP	City
ECLIPSE FOUNDATION	Eclipse Foundation Inc.	K2G 6B1	Ottawa
ECLIPSESOURCE MÜNCHEN	EclipseSource München GmbH	80992	München

Company: eclipse

☐ Own

Short name * ECLIPSE FOUNDATION

Name * Eclipse Foundation Inc.

Company no.

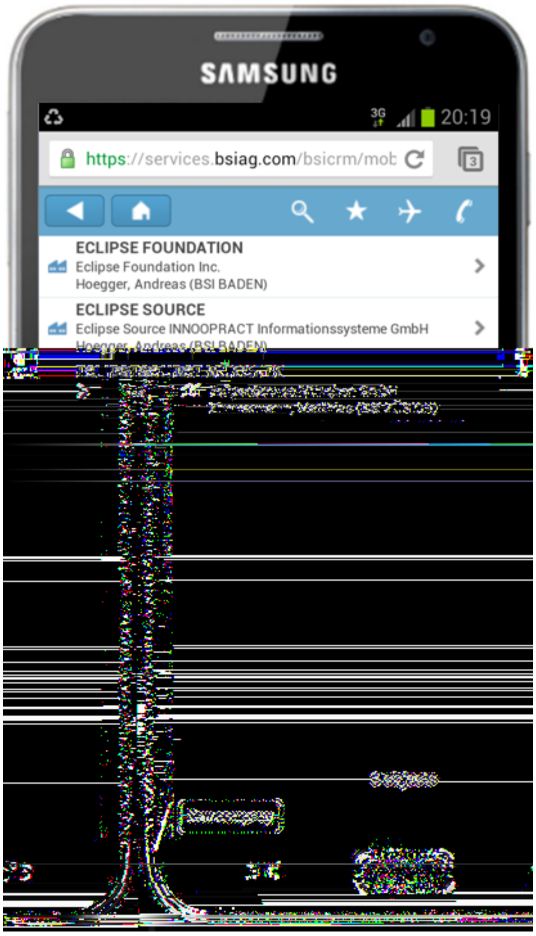
Main representative * Hoegger, Andreas

☑ Active

Region * America

Company type ☑ Influencer

Figure 1.2: A Scout enterprise application running in a web browser.



ECLIPSE FOUNDATION
Eclipse Foundation Inc.
Hoegger, Andreas (BSI BADEN)

ECLIPSE SOURCE
Eclipse Source INNOOPRACT Informationssysteme GmbH

Figure 1.3: The same Scout enterprise application running on a mobile device.

easily be provided for Scout business applications. The primary benefits of Scout web applications for end users are a consistent look and feel and behaviour for both the web client and the desktop client. In Figure 1.2 a screenshot of the web client of the CRM Scout application is shown.

Often, user find themselves in meetings or on the move where it is not possible to work with desktop or laptop computers. In such situations, accessing applications via tablets or mobile phones is the only meaningful approach. As shown in Figure 1.3, a screenshot of the familiar CRM application is provided. The user still benefits from the familiar look and feel and the known functionality of the application.

In contrast to desktop and web applications, most tablets and mobile phones are controlled using touch features instead of mouse clicks. In addition, less elements may be presented on a single screen compared to desktop devices. These two aspects makes it impractical to directly reuse the desktop user interface on mobile devices. Using the Scout mobile extension, the user interface of Scout applications can be transformed on the fly to adapt to touch control and the smaller form factor of mobile devices. This transformation can be observed when comparing the company table shown in the background of Figure 1.1 with the company list presented in Figure 1.3. The multi-column table of the desktop client has been transformed into a list providing the content of the

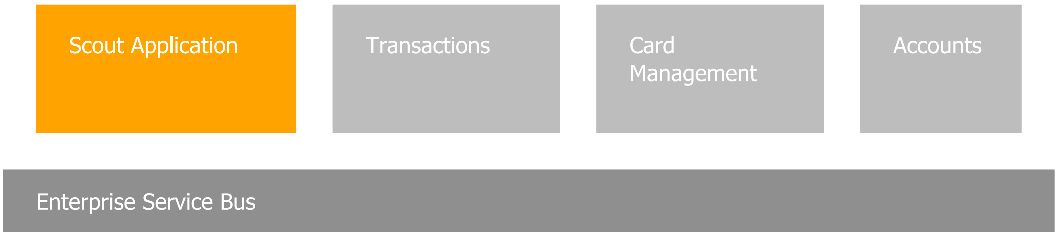


Figure 1.4: A typical application landscape including a service bus and a Scout application.

first few table columns on separate rows. In addition, the context menu ”New company” is now provided as a touch button. As the navigation in the application and the offered choices remain the same for Scout desktop and mobile applications, the end user feels immediately comfortable working with Scout mobile applications.

1.1.2 Management Perspective

For the management, Scout is best explained in terms of benefits it brings to the organisation in question. This is why we are going to concentrate on a (typical) application migration scenario here. Let us assume that to support the company's business, a fairly large landscape of multi-tier applications has to be maintained and developed. Including host systems, client server applications including desktop clients, as well as newer web based applications.

Usually, these applications interact with each other through a service bus as shown in Figure 1.4. Often, some of the applications that are vital to the organisation's core business have grown historically and are based on legacy technologies. And for technologies that are no longer under active development it can get difficult to find staff having the necessary expertise or motivation. Sometimes, the organisation is no longer willing to accept the costs and technology risks of such mission critical applications.

In this situation, the company needs to evaluate if it should buy a new standard product or if the old application has to be migrated to a new technology stack. Now let us assume, that available products do not fit the company's requirements well enough and we have to settle for the migration scenario. In the target architecture, a clean layering similar to the one shown in Figure 1.5 is often desirable.

While a number of modern and established technologies exist that address the backend side (data bases, data access and business services), the situation is different for the UI layer and the application layer. The number of frameworks to develop web applications with Java is excessively large[1], but the choice between desktop application technologies in the Java domain is restricted to three options only. Swing, SWT and JavaFX. Both Eclipse SWT and Java Swing are mature and well established but Swing is moving into 'maintenance only' mode and will be replaced by JavaFX. However, the maturity of the new JavaFX technology in large complex enterprise applications is not yet established. Obviously, deciding for the right UI technology is a challenge and needs to be made very carefully. Reverting this decision late in a project or after going into production can get very expensive and time consuming.

Once the organisation has decided for a specific UI technology, additional components and

---

[1] Web application framework comparison: `http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks#Java`.

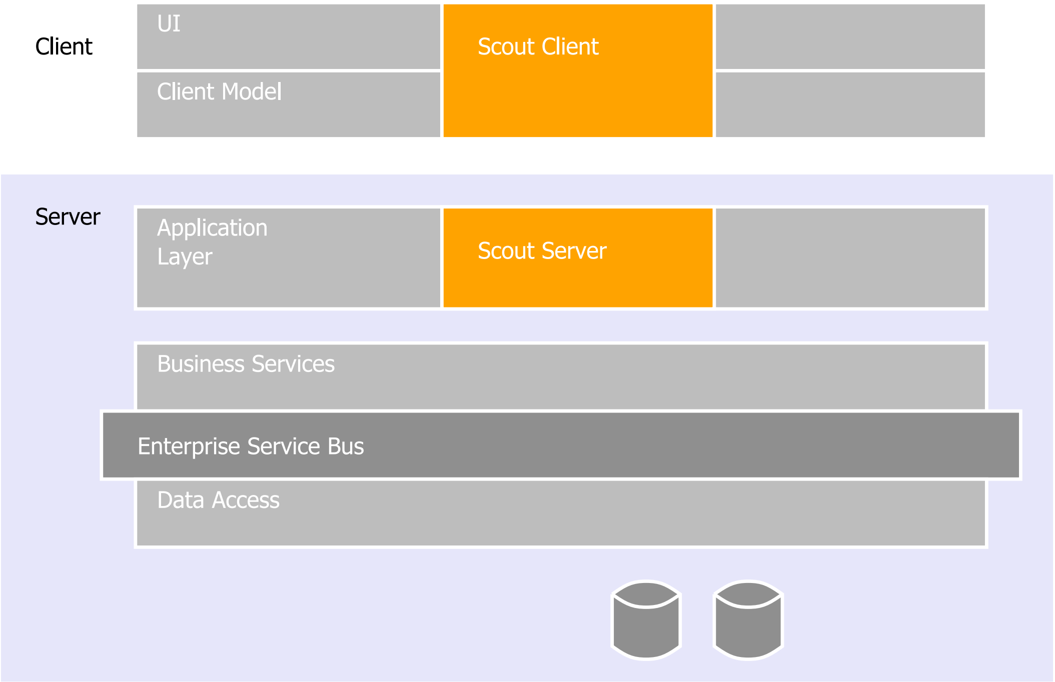


Figure 1.5: The integration of a Scout application in a typical enterprise setup.

frameworks need to be evaluated to cover client server communication, requirements for the application layer, and integration into the existing application landscape. To avoid drowning in the integration effort for all the elements necessary to cover the UI and the application layer a 'lightweight' framework is frequently developed. When available, this framework initially leads to desirable gains in productivity. Unfortunately, such frameworks often become legacy by themselves. Setting up a dedicated team to actively maintain the framework and adapt to new technologies can reduce this risk. But then again, such a strategy is expensive and developing business application frameworks is usually not the core business of a company.

Can we do better? To implement a business application that covers the UI and the application layer as shown in Figure 1.5, Eclipse Scout substantially reduces both risk and costs compared to the inhouse development presented above. First or all, Scout is completely based on Java and Eclipse. Chances are, that developers are already familiar with some of these technologies. This helps in getting developers up to speed and keeping training costs low.

On the UI side, Scout's multi-frontend support almost allows to skip the decision for a specific UI technology. Should a particular web framework become the de-facto standard in the next years, it will be the responsibility of the Scout framework to provide the necessary support. Existing Scout applications can then switch to this new technology with only minimal effort. This is possible because the Scout developers are designing and building the UI of an application using Scout's client model. And this client model is not linked to any specific UI technology. Rather, specific UI renderers provided by the Scout framework are responsible to draw the UI at runtime.

As Scout is an open source project, no licence fees are collected. Taking advantage of the growing popularity of Scout, free community support is available via a dedicated forum. At the same time, professional support is available if the organisation decides for it.

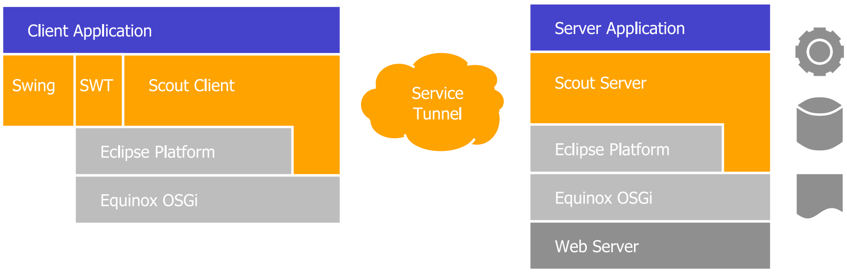


Figure 1.6: The architecture of a Scout client server application including a desktop client.

Figure 1.7: The architecture of a Scout web/mobile client server application.

As the migration of aging applications to current technology is always a challenge, it surely helps to have Scout in the technology portfolio. Not only is it a low risk choice, but also boosts developer productivity and helps to motivate the development team.

1.1.3 Developer Perspective

From the perspective of application developers, Scout offers a Java based framework that covers the complete client server architecture. This implies that – once familiar with the Scout framework – the developer can concentrate on a single framework language (Java) and a single set of development tools.

As Scout is completely based on Java and Eclipse, Scout developers can take full advantage of existing knowledge and experience in these domains. And to make learning Scout as simple as possible, Scout includes a comprehensive software development kit (SDK), the Scout SDK. The Scout SDK helps to create a robust initial project setup for client server applications and includes a large set of wizards for repetitive and error prone tasks. The architecture of a Scout client server created by the Scout SDK is shown in In Figure 1.6.

On the client-side Scout's flexible client model allows the developer to create a good user experience without having to care about specific UI technologies. The reason for this can be found in Scout's client architecture that cleanly separates the UI model from the UI technology. This is illustrated in Figure 1.6 and Figure 1.7 where the Scout UI rendering compontents for Swing, SWT and RAP are shown to be part of the Scout framwork itself. The price of this architecture is the fact, that in Scout (almost) every UI component is implemented four times. First the implementation of the UI model component and then, three rendering components for each UI technology supported by Scout. For desktop clients these are the Swing and the SWT technologies, and for the web and mobile support this is Eclipse RAP which in turn takes care of the necessary JavaScript parts.

Not having to worry about Swing, SWT or JavaScript can significantly boost the productivity. With one exception. If a specific UI widget is missing for the user story to be implemented, the Scout developer first needs to implement such a widget. Initially, this task is slightly more complex than not working with Scout. For custom widgets the Scout developer needs to implement both a model component and a rendering component for a specific UI technology. But as soon as the client application needs to be available on more than a single frontend, the investment already pays off. The developer already did implement the model component and only needs to provide an additional rendering component for the new UI technology. In most situations the large set of Scouts UI components provided out-of-the box are sufficient and user friendly applications are straight forward to implement. Even if the application needs to run on different target devices simultaneously.

Client-server communication is an additional aspect where the developers is supported by Scout. Calling remote services in the client application that are provided by the Scout server looks identical to the invocation of local services. The complete communication including the transfer of parameter objects is handled fully transparent by the Scout framework. In addition, the Scout SDK can completely manage the necessary transfer objects to fetch data from the Scout server that is to be shown in dialog forms on the Scout client. The binding of the transferred data to the form fields is done by the framework.

Although the Scout SDK wizards can generate a significant amount of code, there is no one-way code generation and no meta data in a Scout application. Just the Java code. Developers preferring to write the necessary code manually, may do so. The Scout SDK parses the application's Java code in the background to present the updated Scout application model to the developers preferring to work with the Scout SDK.

Finally, Scout is an open source framework hosted at the Eclipse foundation. This provides a number of interesting options to developers that are not available for closed source frameworks. First of all, it is simple to get all the source code of Scout and the underlying Eclipse platform. This allows for complete debugging of all problems and errors found in Scout applications. Starting from the application code, including the Scout framework, Eclipse and down to the Java platform.

Scout developer can also profit from an increasing amount of free and publicly available documentation, such as this book or the Scout Wiki pages. And problems with Scout or questions that are not clearly addressed by existing documentation can be discussed in the Scout forum. The forum is also a great place for Scout developers to help out in tricky situation and learn from others. Ideally, answered questions lead to improved or additional documentation in the Scout Wiki.

At times, framework bugs can be identified from questions asked in the forum. As all other enhancement requests and issues, such bugs can be reported in Bugzilla by the Scout developer. Using Bugzilla, Scout developers can also contribute bug analysis and patch proposals to solve the reported issue. With this process, Scout developers can actively contribute to the code base of Eclipse Scout. This has the advantage, that workarounds in existing Scout applications can be removed when an upgrade of the Scout framework is made.

Having provided a significant number of high quality patches and a meaningful involvement in the Scout community, the Scout project can nominate a Scout developer as a new Scout committer. Fundamentally, such a nomination is based on the trust of Scout committers in the candidate. To quote the official guidelines[2] for nominating and electing a new committer:

---

[2] Nominating and electing a new Eclipse Scout committer: `http://wiki.eclipse.org/Development_Resources/HOWTO/Nominating_and_Electing_a_New_Committer#Guidelines_for_Nominating_and_Electing_a_New_Committer`.

> *A Committer gains voting rights allowing them to affect the future of the Project. Becoming a Committer is a privilege that is earned by contributing and showing discipline and good judgment. It is a responsibility that should be neither given nor taken lightly, nor is it a right based on employment by an Eclipse Member company or any company employing existing committers.*

After a successful election process (existing committers voting for and not against the candidate) the Scout developer effectively becomes a Scout committer. With this new status, the Scout developer then gets write access to the Eclipse Scout repositories and gains voting rights and the possibility to shape the future of Scout.

1.2 Why should I choose Scout?

Below a number of facts around the background and usage of Scout applications is provided in the form of bullet points. And for each of the statements it would be possible to write a story on its own. But instead, we just leave the sentences in the format of an elevator pitch.

1.2.1 Get up to speed in days

- Scout is based on Java
- Any Java developer quickly learns Scout
- The Scout SDK provides great tooling

1.2.2 Run your apps on web, desktop and mobile

- Scout developers implement the application model only
- Choose the UI technology independently of your application
- For desktop apps, choose SWT or Swing
- For browser or mobile apps, choose RAP

1.2.3 Scout is production ready

- Scout has a 10 years history of being used in production
- Scout is used for dozens of product installations, some with thousands of users
- Scout applications are running across industries: finance, transportation, government, retail

1.2.4 Protect your investment

- Scout is an open source project hosted at the Eclipse foundation
- Scout has an active and growing community
- Scout is moving with the best available technologies.

1.3 What should I read?

The text below provides guidelines on what to read (or what to skip) depending on your existing background.

We first address the needs of junior Java developers that like to learn more about developing enterprise applications. Then, we suggest a list of sections relevant for software wizards that already have a solid understanding of the Eclipse platform, Java enterprise technologies, and real world applications. Finally, the information needs of IT managers are considered.

1.3.1 I know some Java

The good news first. This book is written for you! No prior knowledge of the Eclipse Platform[3] is needed. We do not even assume that you have a meaningful understanding of the Java Enterprise Edition (Java EE)[4]. Of course, having prior experience in client server programming with Java is helpful. It also helps having used the Eclipse IDE for Java development before — please do not mistake the IDE with the Eclipse platform[5]. However, prior knowledge of Java EE and the Eclipse platform is not required for this book.

The "bad" news is, that writing Scout applications requires a solid understanding of Java. To properly benefit from this book, we assume that you have been developing software for a year or more. And you should have mastered the Java Standard Edition (Java SE)[6] to a significant extent. To be more explicit, you are expected to be comfortable with all material required for the Java Programmer Level I Exam[7] and most of the material required for Level II[8].

Luckily, free online tutorials to learn Java are offered in many places. A good starting point is the official Java documentation site[9]. If you prefer to work with video tutorials we recommend "Eclipse and Java for Total Beginners"[10], although the installation used is somewhat out of date. As for printed books, we suggest to start with either "Head First Java"[1] or "Thinking in Java"[3]. Highly recommended but slightly more advanced is "Effective Java"[2].

To solve the really tricky Java problems there is often no way around the Java specification[11] itself. Just make sure to pick the right Java version for your context.

We now propose to start downloading and installing Scout as described in Appendix B and do some actual coding. To do so, please continue with the "Hello World" example provided in Chapter 2. You can expect to complete this example in less than two hours including the necessary download and installation steps. Afterwards, you might want to continue with the remaining material in "Getting Started". Working through the complete "Getting Started" should take no more than two days. This exercise will provide you with a broad overview of Eclipse Scout and enough hands-on material to decide how much Scout will help you with your current and future projects.

Once you work with the Scout framework on a regular basis, you might want to ask questions in the Scout forum[12]. When your question gets answered, please ask yourself if your initial problem could have been solved by better documentation. In that case, you might want to help the Scout community by fixing or amending the Scout wiki pages[13]. Or this book. If you find a bug in Eclipse Scout that makes your life miserable you can report it. When your bug is fixed, you can test the fix. To help speed up the bug fixing process you can contribute patches. All of these actions will add to the healthy grow of the Scout community. And this is exactly the topic of "Contributing", the last part of this book.

---

[3] Eclipse Platform: `http://wiki.eclipse.org/Platform`

[4] Java Enterprise Edition: `http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition`

[5] By reading through the book you will learn that there is much more to the Eclipse platform than just the IDE

[6] Java Standard Edition: `http://en.wikipedia.org/wiki/Java_SE`

[7] Level I Exam: `docs.oracle.com/javase/tutorial/extra/certification/javase-7-programmer1.html`

[8] Level II Exam: `docs.oracle.com/javase/tutorial/extra/certification/javase-7-programmer2.html`

[9] Official online Java tutorial: `http://docs.oracle.com/javase/tutorial/`

[10] Eclipse and Java for Total Beginners: `http://eclipsetutorial.sourceforge.net/totalbeginner.html`

[11] The Java Language Specification `http://docs.oracle.com/javase/specs/`

[12] Eclipse Scout forum: `http://www.eclipse.org/forums/eclipse.scout`

[13] Eclipse Scout wiki: `http://wiki.eclipse.org/Scout`

1.3.2 I know tons of both Java and Eclipse

This means that you are one of these software wizards that get easily bored. You probably hate going through lengthy descriptions of widely known concepts. In that case let us assume that you are prepared to spend two hours to grasp the scope of Eclipse Scout and get an impression of its strengths and limitations. The list below suggests a sequence of sections to digest including a brief motivation for each one.

- Chapter 2 ”Hello World” Tutorial. Download and installation of the Scout package should take less than 30 minutes, going through the ”Hello World” takes another 15 minutes.
- Section 3.2 “Walking through the Initial Application” Read about some key elements used in every Scout client application including integration of server services and data binding.
- Chapter 4 “Scout Tooling”. Browse through the tooling chapter to get an impression on the tooling provided with Scout. Make sure you understand that the Scout SDK is supporting the developer without restricting the developer.
- Section 5.1 “The My Contacts Application”. Check out the slightly larger demo application. In case you are not yet running out of time, download the demo app as described in the Scout wiki[14].

1.3.3 I am a manager

Being a manager and actually reading this book may indicate one of the following situations:

- Your developer tried to convince you that Eclipse Scout can help you with implementing business applications in a shorter time and for less money. And you did not understand why (again) a new technology should work better than the ones you already use.
- Your are a product manager of a valuable product that is based on legacy technology. And you are now evaluating options to modernize your product.
- Fill in your current situation. There must be a reason why you are checking out this book.

If you did not already go through Section 1.1 please do so now. Then, flip through Section 5.1 to get an impression of the ”My Contacts” application. In case you like the idea that your developers should be able to build such an application in a single day, you might want to talk to us[15].

---

[14] Download and installation of the ”My Contacts” application: `http://wiki.eclipse.org/Scout/Book/3.9#Download_and_Run_the_Scout_Sample_Applications`.

[15] To contact the Scout team, use the email provided on the Scout homepage: `http://eclipse.org/scout/support/`.

Chapter 2

”Hello World” Tutorial

The ”Hello World” chapter walks you through the creation of an Eclipse Scout client server application. When the user starts the client part of this application, the client connects to the server[1] and asks for some text content that is to be displayed to the user. Next, the server retrieves the desired information and sends it back to the client. The client then copies the content obtained from the server into a text field widget. Finally, the client displays the message obtained from the server in a text field widget.

The goal of this chapter is to provide a first impression of working with the Scout framework using the Scout SDK. We will start by building the application from scratch and then we'll deploy the complete application to a Tomcat web server. Except for a single line of code in the server part of the ”Hello World” application, we will only be using the tooling provided by the Scout SDK.

Based on this simple ”Hello World” applications a large number of Scout concepts can be illustrated. Rather than including this background material in this tutorial, such information is provided separately in Chapter 3. This tutorial is also available in the Scout wiki[2].

2.1 Installation and Setup

Before you can start with the ”Hello World” example you need to have a complete and working Scout installation. For this, see the step-by-step installation guide provided in Appendix B. Once you have everything installed, you are ready to create your first Scout project.

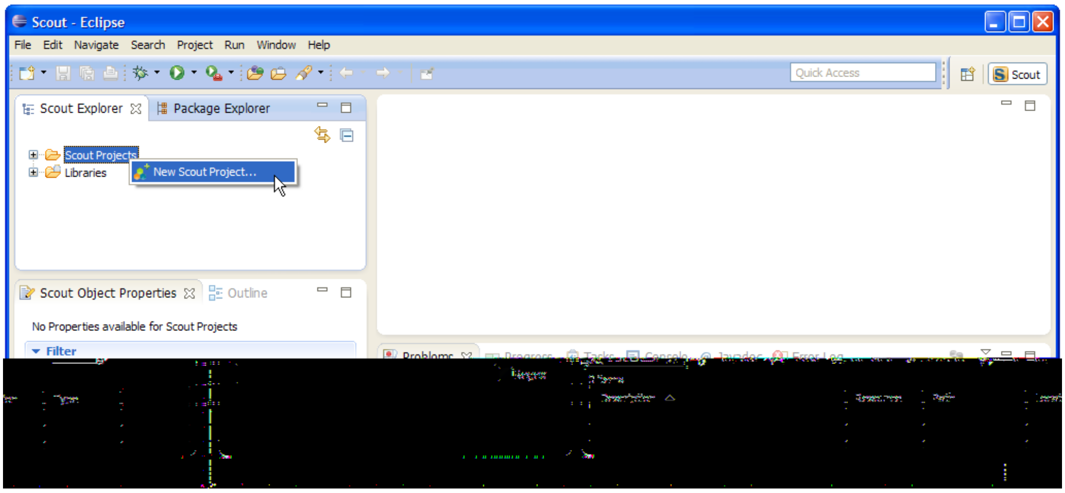
2.2 Create a new Project

Start your Eclipse IDE and select an empty directory for your workspace. This workspace directory will then hold all the project code for the ”Hello World” application. In the started Eclipse IDE, you can then create a new Scout project by selecting the New Scout Project... context menu as shown in Figure 2.1. Alternatively, you can also use the Eclipse File—New—Project... menu. In this case you are shown the generic *New Project* wizard of Eclipse where you can select

---

[1] The Scout server part of the ”Hello World” application will be running on a web server.

[2] ”Hello World” wiki tutorial: `http://wiki.eclipse.org/Scout/Tutorial/3.9/HelloWorld`



No Properties available for Scout Projects

Figure 2.1: Create a new Scout project using the Scout SDK perspective.

the *Scout Project* wizard below the Scout folder as seen in Figure 2.2. With the NEXT button you will be directed to the next dialog step, the *New Scout Project* wizard.

In the *New Scout Project* wizard enter a name for you Scout project. As we are creating a "Hello World" application, use `org.eclipsescout.helloworld` for the *Project Name* field. Now, click the FINISH button to let the Scout SDK create the initial project code for you.

Once the initial project code is built, the Scout SDK displays the application model in the *Scout Explorer*. This model is visually presented as a tree structure covering both the client and the server part of the application. In Figure 2.3 the Scout Explorer displays the top level elements of the complete Scout application.

2.3 Run the Initial Application

After the initial project creation step we are ready to start the server and the clients of the still empty Scout application. For this, we switch to the Scout Explorer and select the root node *org.eclipse.scout.helloworld*. Selecting the application's *org.eclipse.scout.helloworld* node in the Scout Explorer displays the product launchers in the *Scout Object Properties*. As we can see in Figure 2.4, we have product launchers for four different development products.

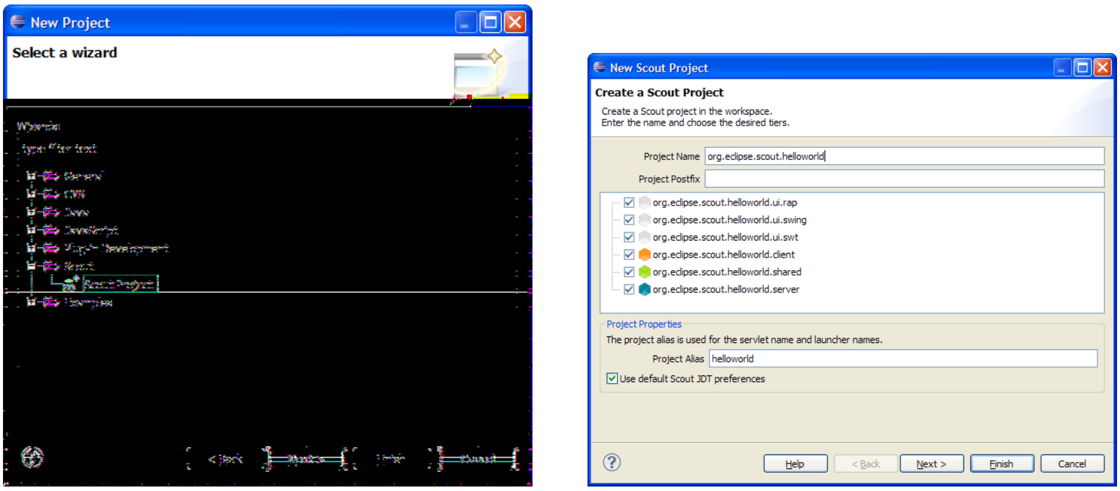
RAP	The RAP server application for web and mobile clients
Swing	The Scout Swing desktop client application
SWT	The Scout SWT desktop client application
Server	The Scout server application

Each product launcher box provides a link to the corresponding Eclipse product file[3], the configuration file[4], as well as three launcher icons to start and stop the corresponding application. The green CIRCLE icon starts the product in normal mode. The BUG icon just below, starts a product in debug mode. To terminate a running product, the red SQUARE icon is provided.

---

[3] Product files define all the necessary elements of an application.

[4] The configuration file `config.ini` provides parameters that are read at startup of the corresponding program.



Create a Scout Project

Create a Scout project in the workspace.
Enter the name and choose the desired tiers.

Project Name: org.eclipse.scout.helloworld
Project Postfix:

☑ org.eclipse.scout.helloworld.ui.rap
☑ org.eclipse.scout.helloworld.ui.swing
☑ org.eclipse.scout.helloworld.ui.swt
☑ org.eclipse.scout.helloworld.client
☑ org.eclipse.scout.helloworld.shared
☑ org.eclipse.scout.helloworld.server

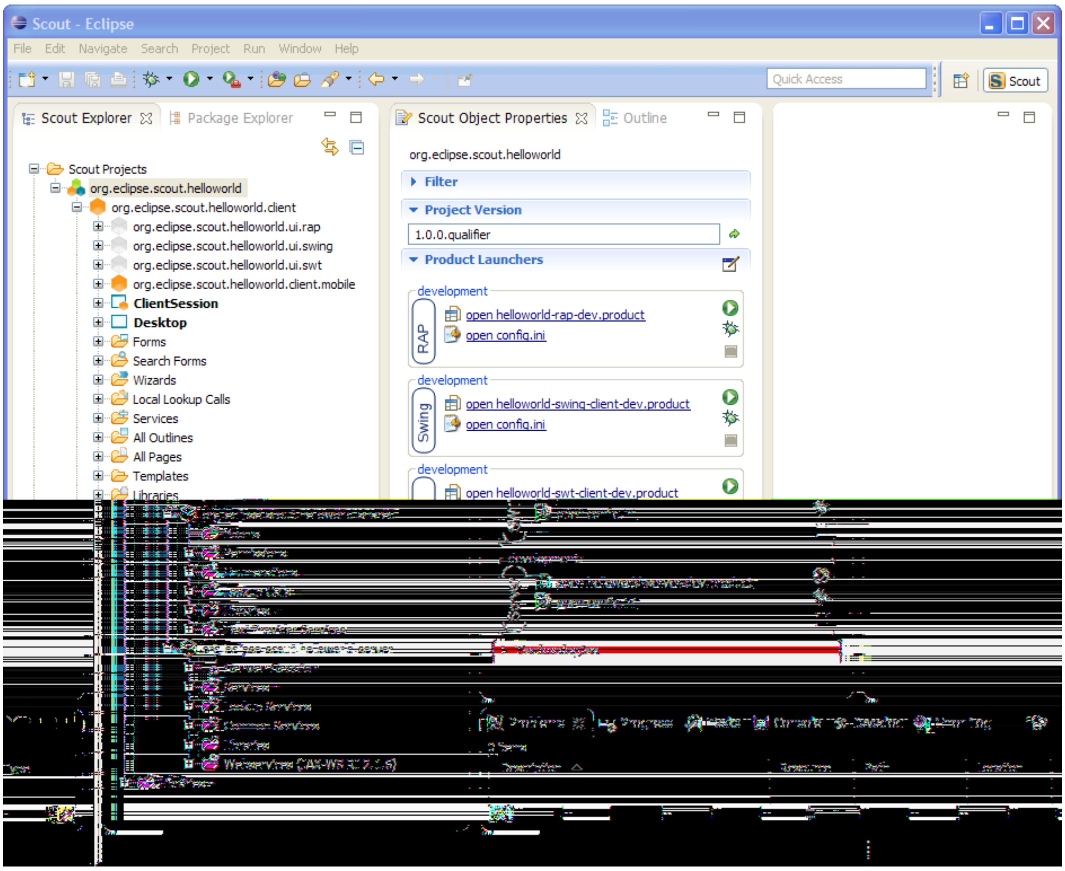
Project Properties

The project alias is used for the servlet name and launcher names.

Project Alias: helloworld

☑ Use default Scout JDT preferences

Figure 2.2: The new project wizard. The dialog on the left side is only shown when using the generic *New Project* wizard of Eclipse



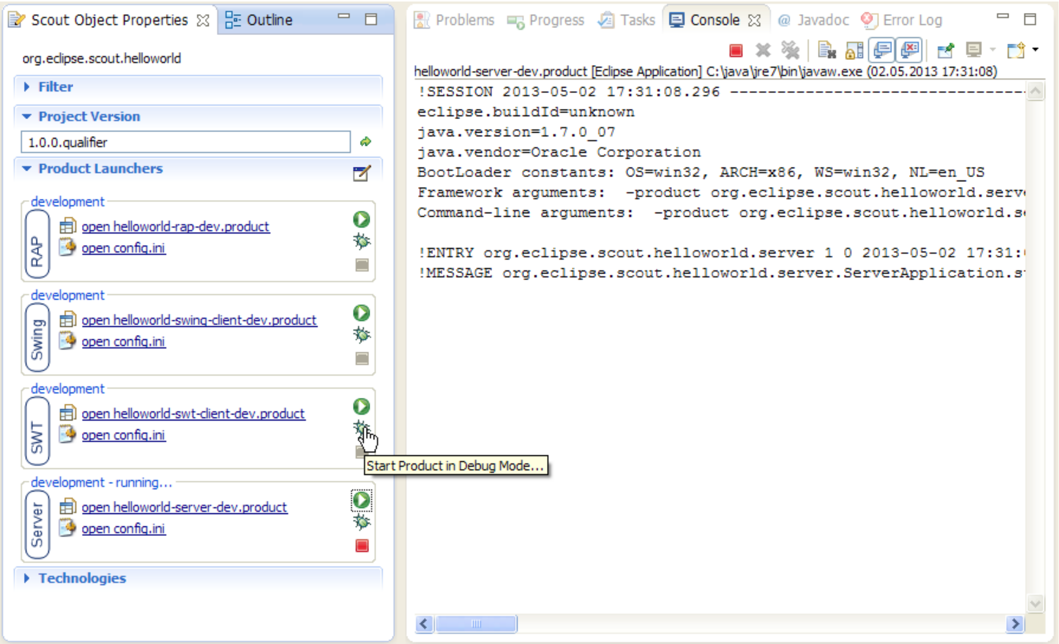
Project Version: 1.0.0.qualifier

development (RAP): open helloworld-rap-dev.product, open config.ini

development (Swing): open helloworld-swing-client-dev.product, open config.ini

development: open helloworld-swt-client-dev.product

Figure 2.3: The Scout SDK showing the tree representation of our "Hello World" application in the Scout Explorer. Scout Object Properties displays the product launchers for the server and the available clients.



1.0.0.qualifier

```
helloworld-server-dev.product [Eclipse Application] C:\java\jre7\bin\javaw.exe (02.05.2013 17:31:08)
!SESSION 2013-05-02 17:31:08.296 --------------------------------
eclipse.buildId=unknown
java.version=1.7.0_07
java.vendor=Oracle Corporation
BootLoader constants: OS=win32, ARCH=x86, WS=win32, NL=en_US
Framework arguments:  -product org.eclipse.scout.helloworld.serv
Command-line arguments:  -product org.eclipse.scout.helloworld.s

!ENTRY org.eclipse.scout.helloworld.server 1 0 2013-05-02 17:31:
!MESSAGE org.eclipse.scout.helloworld.server.ServerApplication.s
```

Figure 2.4: Starting the SWT client in the Scout SDK using the provided SWT product launcher. Make sure to start the server before starting any client product.

Alternatively, you can also stop products by clicking on the same red icon in the console view. This is shown on the right hand side of Figure 2.4. Client products may also be stopped by closing the client's main window or using the provided FILE—EXIT menu.

Before any of the client products is started, we need to start the server product using the green circle or the bug launcher icon. During startup of the Scout server you should see console output similar to the one shown on the right hand side of Figure 2.4. Once the server is running, you may start the SWT client as shown in Figure 2.4, the Swing client, and the RAP server in the same way. With a running RAP product, you may also display the Scout client in a web browser. Just type the address `http://localhost:8082/web` into the browser's navigation bar.

Having started the Scout server and all client products, the client applications should become visible as shown in Figure 2.5.

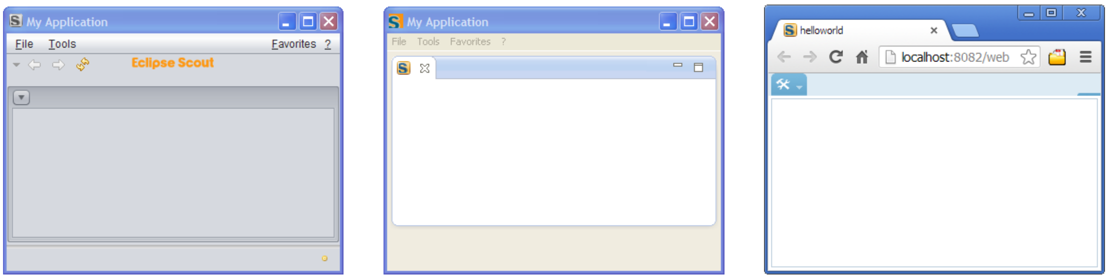


Figure 2.5: Running the three client applications. Each client displays an empty desktop form. From left to right: The Swing client, the SWT client, and the web client

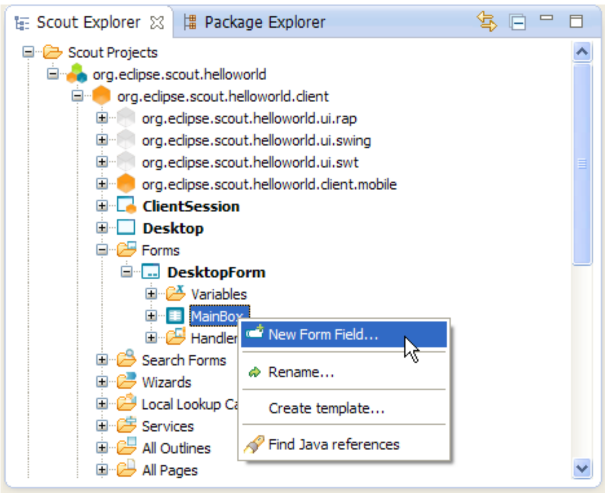


Figure 2.6: Using the New Form Field ... menu to start the form field wizard provided by the Scout SDK.

2.4 The User Interface Part

In this section we will define the user interface (UI) model for the "Hello World" application. More specifically, we add a text field widget to the client's empty desktop form of the "Hello World" application. In the steps described below, we use the *New Form Field* wizard provided by the Scout SDK. First, we add a group box field to the desktop form. Then, we apply this wizard a second time to add the actual text widget[5].

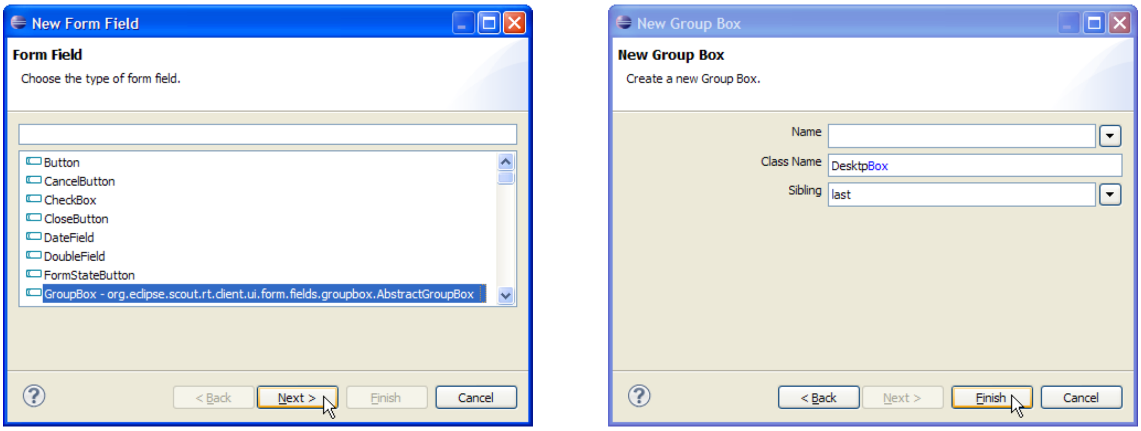
To add any widgets to the desktop form we first need to navigate to the *DesktopForm* in the Scout Explorer. By clicking on the small plus icon on the left hand side of the *DesktopForm* this element is expanded and the *MainBox* element becomes visible below. With a click of the right mouse button over the *MainBox*, the available context menus are displayed. To start the form field wizard we select the New Form Field ... menu as shown in Figure 2.6.

In the first dialog of the form field wizard shown on the left side of Figure 2.7, we choose the form field type. To select the desired field type, we either select the desired type with the mouse or use the search field to filter the list of available field types. In the second wizard dialog, we do not provide a label for the group box in the *Name* field. As we have only a single group box in the "Hello World" desktop form we omit the name and enter 'Desktop' into the *Class Name* field before we close the wizard with the Finish button. This step is shown on the right side of Figure 2.7. The Scout SDK will add the necessary Java code for the `DesktopBox` in the background.

To add the text field widget to the group box just created, we navigate to the corresponding *DesktopBox* in the Scout Explorer. For this, we click on the small plus icon on the left hand side of the *MainBox* to expand this node and make the *DesktopBox* element visible. On *DesktopBox* we again use the New Form Field ... menu. In the first wizard dialog, we select *StringField* using 'st' as a search criteria and click the Next button to open the second wizard dialog. In the second wizard dialog, we enter 'Message' into the *Name* field. As we do not yet have the text 'Message' available in our "Hello World" application the wizard prompts the user with the proposal New

---

[5] Adding top level group boxes to a form helps in structuring the form layout, especially when a form contains a large number of form fields.


New Form Field

Form Field

Choose the type of form field.

- Button
- CancelButton
- CheckBox
- CloseButton
- DateField
- DoubleField
- FormStateButton
- GroupBox - org.eclipse.scout.rt.client.ui.form.fields.groupbox.AbstractGroupBox

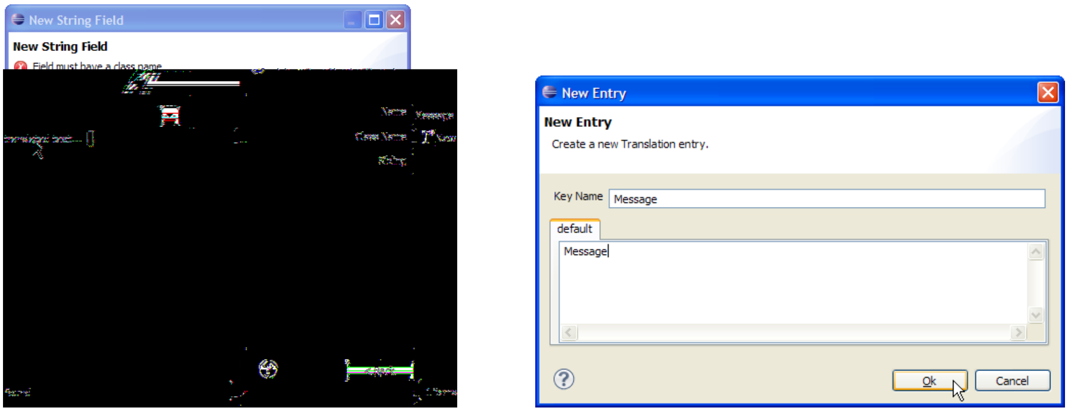
New Group Box

New Group Box

Create a new Group Box.

Name
Class Name: DesktpBox
Sibling: last

Figure 2.7: Adding the *DesktopBox* field with the Scout SDK form field wizard.


New String Field

New String Field

Field must have a class name.

New Entry

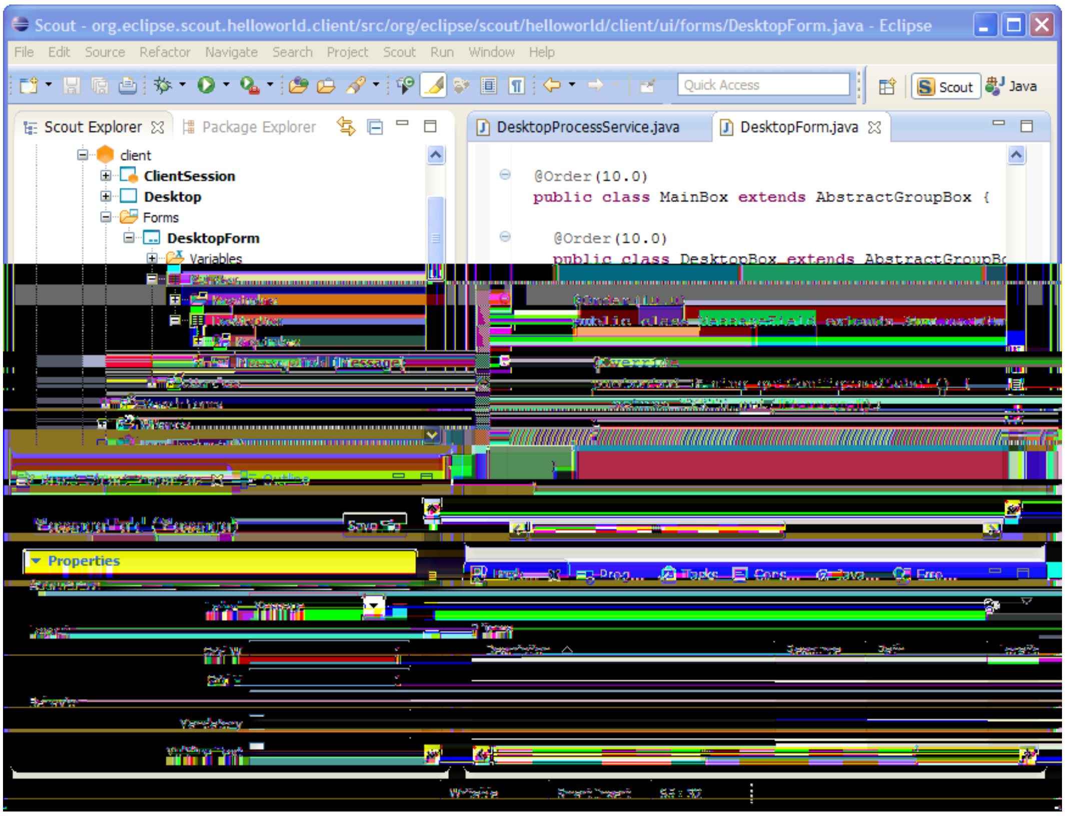
New Entry

Create a new Translation entry.

Key Name: Message

default: Message

Figure 2.8: Adding a *StringField* and providing a new translation entry.


Scout - org.eclipse.scout.helloworld.client/src/org/eclipse/scout/helloworld/client/ui/forms/DesktopForm.java - Eclipse
File Edit Source Refactor Navigate Search Project Scout Run Window Help
Quick Access
Scout
Java
Scout Explorer
Package Explorer
client
ClientSession
Desktop
Forms
DesktopForm
Variables
Properties
DesktopProcessService.java
DesktopForm.java

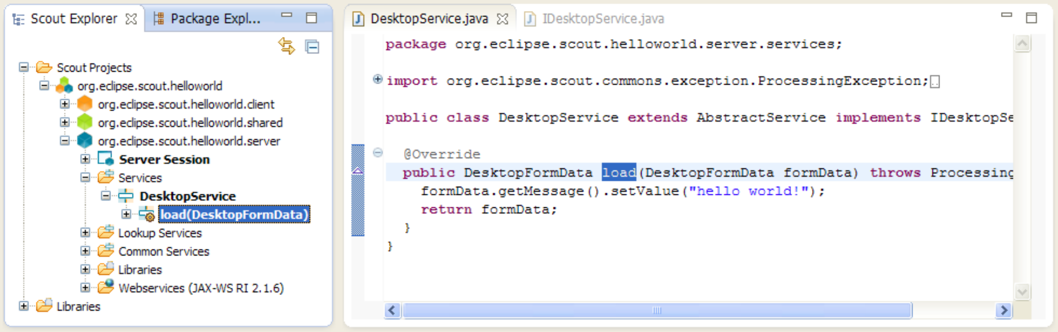
```
@Order(10.0)
public class MainBox extends AbstractGroupBox {

  @Order(10.0)
  public class DesktopBox extends AbstractGroupBc
```

Figure 2.9: Scout SDK showing the *MessageField*

Translated Text .... Selecting the provided option we can add a new text entry as shown in Figure 2.8. Once we have provided some initial translation for our message label, we can close the translation dialog with the Ok button. Finally, we close the form field wizard using the Finish button.

By expanding the *DesktopBox* element in the Scout Explorer, the new message field becomes visible. A double click on the message field element then loads the corresponding Java code into an Editor as shown in Figure 2.9. If you are following this tutorial with your own Eclipse Scout installation compare your status with this screenshot. Make sure that the project structure in the Scout Explorer looks as shown in Figure 2.9 and a double click to the *MessageField* both loads the corresponding Java code and displays the message field's properties in the Scout Object Properties.

Having verified your status of the "Hello World" application you can start the application as described in Section 2.3. The client applications will then display your message widget. However, the text widget is still empty, as we did not yet load any initial content into it. This is the topic of the next section where we continue the tutorial with the server part.



```
package org.eclipse.scout.helloworld.server.services;

import org.eclipse.scout.commons.exception.ProcessingException;

public class DesktopService extends AbstractService implements IDesktopSe

  @Override
  public DesktopFormData load(DesktopFormData formData) throws Processing
    formData.getMessage().setValue("hello world!");
    return formData;
  }
}
```

Figure 2.10: The Scout Explorer showing the blue server node expanded with the *Services* folder. In this folder the *load* method of *DesktopService* is selected and its initial implementation is shown in the editor on the right side.

2.5 The Server Part

The responsibility of the server part in our "Hello World" application is to provide an initial text content for the message field in the client's user interface. We implement this behaviour in the `load` method of the server's `DesktopService`. An empty stub for the `load` method and the `DesktopService` service have already been created during the initial project creation step described in Section 2.2. The `DesktopService` represents the server service corresponding to the `DesktopForm` on the client side. This initial setup represents Scout's standard form processing mechanism where client forms and server services typically come in pairs.

Whenever the client's user interface displays a form to the user, the client connects to the server and calls the `load` method of the corresponding server service. We can now add the business logic to the `load` method of the server's `DesktopService` to implement the desired behaviour. And this is the last missing piece to complete the "Hello World" application.

To navigate to the implementation of the desktop service in the Scout SDK, we first expand the blue top-level *server* node in the Scout Explorer. Below the server node, we then expand the *Services* folder which shows the *DesktopService* element. Expanding this *DesktopService* node, the `load` method becomes visible as shown in Figure 2.10.

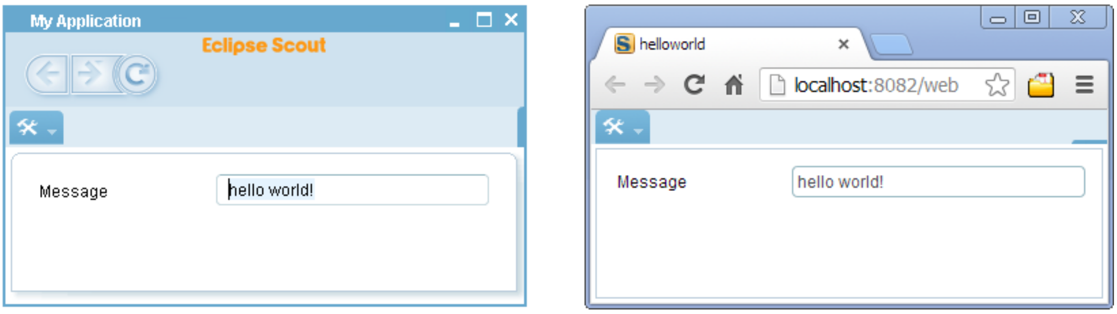
```
@Override
public DesktopFormData load(DesktopFormData formData) throws ↘
  →ProcessingException {
  //TODO [mzi] Auto-generated method stub.
  return formData;
}
```

According to the signature of the `load` method, a `formData` object is passed into this method that is then handed back in the return statement. To complete the implementation of the `load` method it is sufficient to assign the text 'hello world!' to the message field part of the form data. The only line of Java code we write in our "Hello World" application is printed below. For the complete implementation of the load method see Listing 2.1.

```
formData.getMessage().setValue("hello world!");
```

Listing 2.1: Assigning "hello world" to the form data's message field.

```
 @Override
 public DesktopFormData load(DesktopFormData formData) throws ↘
   →ProcessingException {
  formData.getMessage().setValue("hello world!");
  return formData;
 }
}
```



Message: hello world!

Message: hello world!

Figure 2.11: The "Hello World" client application with the Rayo look and feel. The desktop client is shown on the left and the web client on the right hand side.

2.6 Add the Rayo Look and Feel

For Eclipse Scout applications a slick look and feel called Rayo is available in the Eclipse Marketplace[6]. And in this (optional) part of the "Hello World" tutorial we will add Rayo to our "Hello World" Swing client application. As a result, we will get a Scout desktop application that looks the same as the corresponding Scout web client as shown in Figure 2.11.

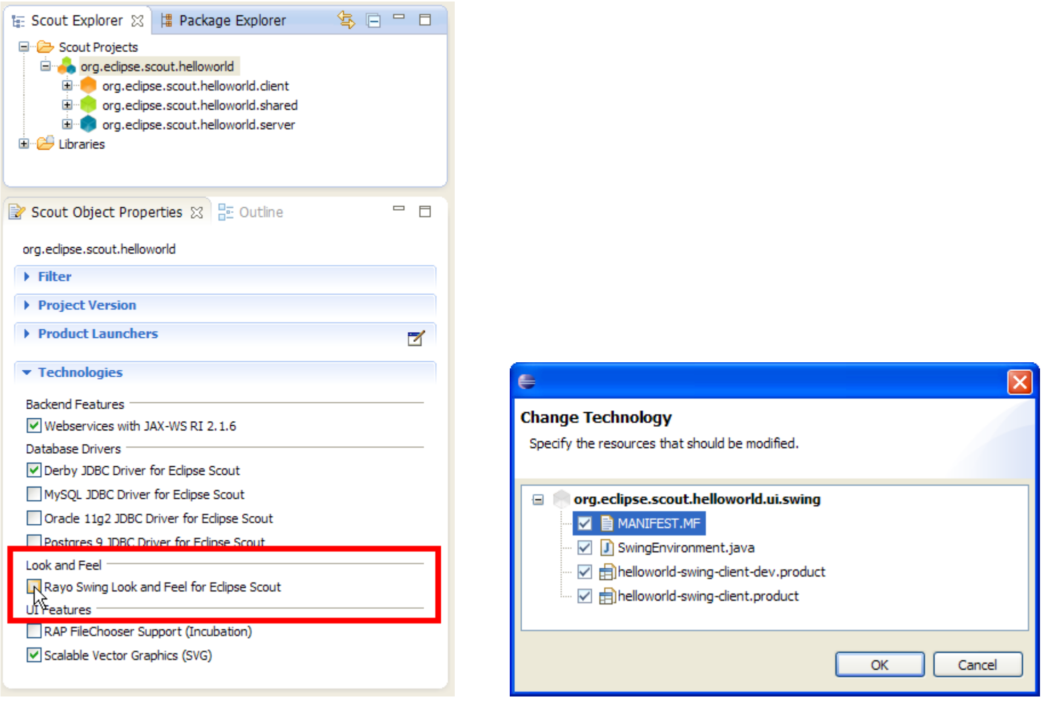
To add Rayo in the Scout SDK to our "Hello World" project, switch to the Scout Explorer and select the top-level *org.eclipse.scout.helloworld* node. Then, according to Figure 2.12, select the checkbox *Rayo Swing Look and Feel for Eclipse Scout* under the *Technologies* section of the Scout Object Properties. This brings up a dialog showing the proposed changes to the Swing plugin of the "Hello World" application. These changes need to be confirmed with the OK button. The first time the user adds the Rayo feature in the Scout SDK, Eclipse needs to download the package from the Eclipse Marketplace. This download and subsequent installation of Rayo will make you to go through the following steps.

1. Accept Licence: GPL with Classpath Exception
2. Accept unsigned content
3. Restart the Eclipse IDE

After the successful download and installation of the Rayo package, start the Swing client using the procedure described in Section 2.3. When we also start the web client of the "Hello World" application using the RAP product launcher, we can compare the result side by side.

---

[6] Eclipse Marketplace: `http://marketplace.eclipse.org/`



Scout Explorer
Package Explorer
Scout Projects
org.eclipse.scout.helloworld
org.eclipse.scout.helloworld.client
org.eclipse.scout.helloworld.shared
org.eclipse.scout.helloworld.server
Libraries
Scout Object Properties
Outline
org.eclipse.scout.helloworld
Filter
Project Version
Product Launchers
Technologies

Backend Features

☑ Webservices with JAX-WS RI 2.1.6

Database Drivers

☑ Derby JDBC Driver for Eclipse Scout

☐ MySQL JDBC Driver for Eclipse Scout

☐ Oracle 11g2 JDBC Driver for Eclipse Scout

☐ Postgres 9 JDBC Driver for Eclipse Scout

Look and Feel

☐ Rayo Swing Look and Feel for Eclipse Scout

UI Features

☐ RAP FileChooser Support (Incubation)

☑ Scalable Vector Graphics (SVG)

Change Technology

Specify the resources that should be modified.

org.eclipse.scout.helloworld.ui.swing

☑ MANIFEST.MF

☑ SwingEnvironment.java

☑ helloworld-swing-client-dev.product

☑ helloworld-swing-client.product

Figure 2.12: Adding the Rayo Swing look and feel. The Rayo checkbox to activate the look and feel is highlighted on the left hand side. The dialog on the right hand side shows the changes in the Swing plugin that will be made by the Scout SDK.

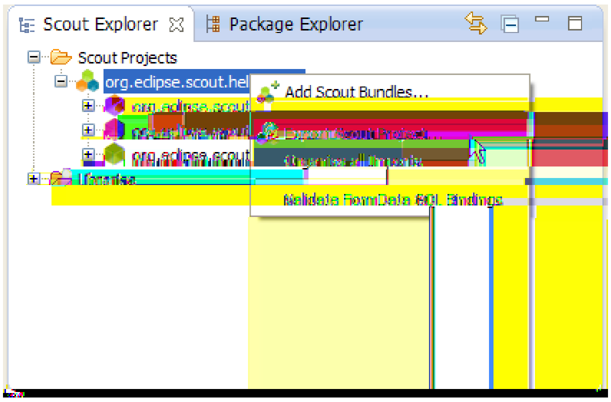
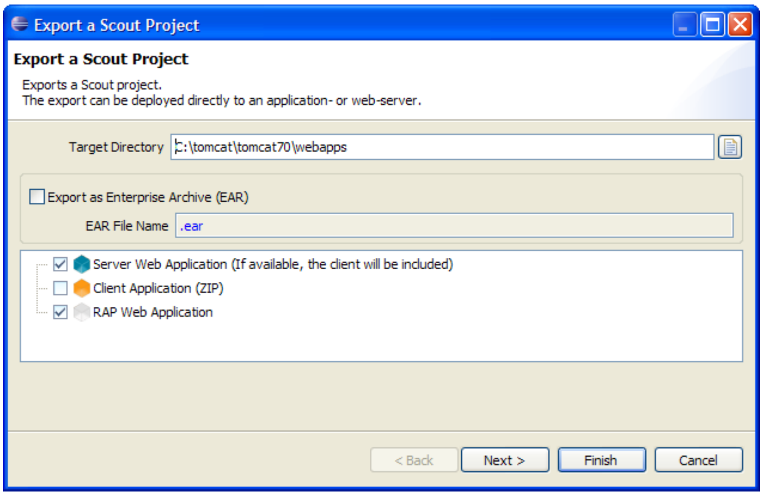


Figure 2.13: Starting the *Export Scout Project* wizard in the Scout SDK with the context menu. In the first wizard step, the target directory for the WAR files and the artefacts to export are specified.



Export a Scout Project

Exports a Scout project.
The export can be deployed directly to an application- or web-server.

Target Directory C:\tomcat\tomcat70\webapps

☐ Export as Enterprise Archive (EAR)

EAR File Name .ear

☑ Server Web Application (If available, the client will be included)
☐ Client Application (ZIP)
☑ RAP Web Application

Figure 2.14: The first dialog of the *Export Scout Project* wizard. Here, the target directory for the WAR files that will be generated by the wizard is specified.

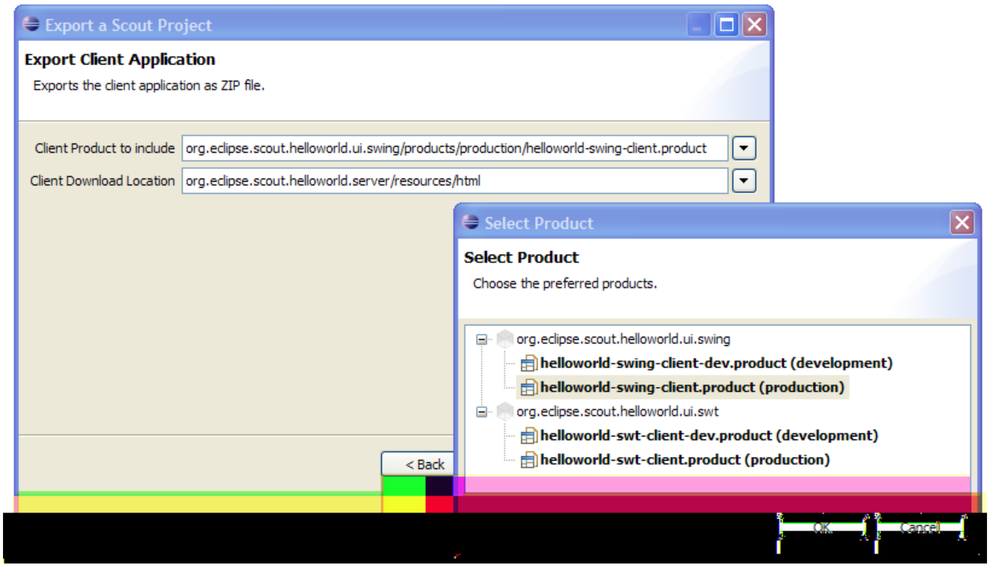
2.7 Exporting the Application

We are now ready to move the finished "Hello World" application from our development environment to a productive setup. The simplest option to move our application into the 'wild' is to use the *Export Scout Project* wizard provided by the Scout SDK. Using the default settings, the export wizard produces two WAR files[7] that contain the complete Scout server and the desktop and mobile client applications.

To deploy the application to a web server the WAR files generated by the wizard are the only artefacts needed. The first WAR file contains the Scout server including a zipped desktop client for downloading. In the second WAR file, the RAP server application that provides both the web client and the client for mobile devices.

---

[7] Web application ARchive (WAR): `http://en.wikipedia.org/wiki/WAR_file_format_(Sun)`



Export Client Application

Exports the client application as ZIP file.

Client Product to include: org.eclipse.scout.helloworld.ui.swing/products/production/helloworld-swing-client.product

Client Download Location: org.eclipse.scout.helloworld.server/resources/html

Select Product

Choose the preferred products.

- org.eclipse.scout.helloworld.ui.swing
  - **helloworld-swing-client-dev.product (development)**
  - **helloworld-swing-client.product (production)**
- org.eclipse.scout.helloworld.ui.swt
  - **helloworld-swt-client-dev.product (development)**
  - **helloworld-swt-client.product (production)**

Figure 2.15: The third dialog of the *Export Scout Project* wizard defines the client application to be included in the `helloworld_server.war` file. In the last step of the export wizard the RAP sever is exported to the specified file name (right).

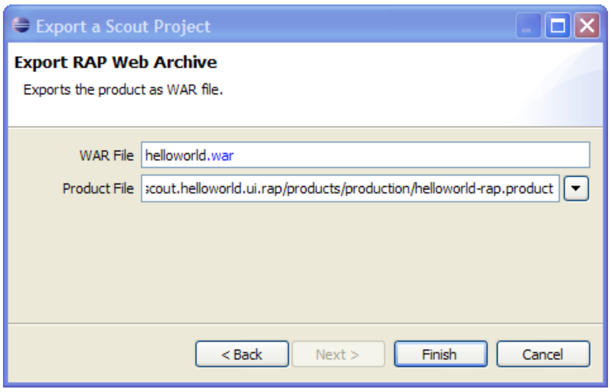
To start the export wizard, we start the Scout SDK with the "Hello World" Scout project. In the Scout Explorer we then select the corresponding EXPORT SCOUT PROJECT... context menu on the "Hello World" top level application node as shown in Figure 2.13. In the first wizard dialog shown in Figure 2.14, the target directory for the WAR files needs to be specified. You may choose any directory as the target directory[8]. After clicking NEXT button the second wizard step proposes the server product file that specifies the artefacts to be exported including the file name for the WAR file for the "Hello World" server application. Typically, the proposed default values are fine. Move to the third dialog with NEXT button.

In the third dialog of the *Export Scout Project* wizard the desktop client to be included in the WAR file needs to be specified. The default selection is set to the SWT client application. For the "Hello World" example, we want to include the Swing client application with the Rayo Look and Feel. For this, we need to change the selected product to *helloworld-swing-client.product (production)* according to Figure 2.15. With NEXT button we move to the last wizard step.

In the last wizard dialog shown in Figure 2.16, the RAP server product and the corresponding WAR file name are specified. Normally, the proposed field values are fine and we can close the wizard with FINISH button. After this last step, the Scout SDK is assembling the necessary artefacts and building the two "Hello World" WAR files. These two WAR files are the only items needed for deploying the "Hello World" application to a web server

---

[8] Make sure to remember the location of this directory. We will need the directory location again when we deploy these WAR files to the Tomcat web server.



Export RAP Web Archive

Exports the product as WAR file.

WAR File: helloworld.war

Product File: ;cout.helloworld.ui.rap/products/production/helloworld-rap.product

Figure 2.16: The last dialog of the *Export Scout Project* wizard defines the export of the RAP server. Normally, the proposed field values do not need any adjustments.

2.8 Deploying to Tomcat

As the final step of this tutorial, we deploy the two WAR files representing our "Hello World" application to a Tomcat web server. For this, we first need a working Tomcat installation. If you do not yet have such an installation you may want to read and follow the instructions provided in Appendix C. To verify a running Tomcat instance, type `http://localhost:8080/` into the address bar of the web browser of your choice. You should then see the page shown in Figure 2.17.

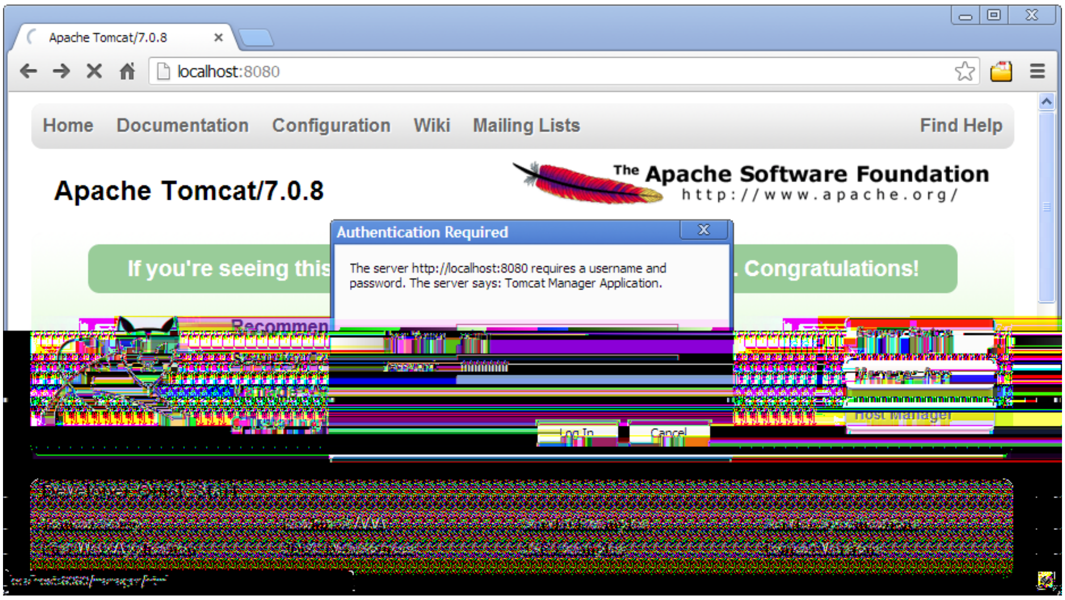
Once the web browser displays the successful running of your Tomcat instance, switch to its "Manager App" by clicking on the button highlighted in Figure 2.17. After entering user name and password the browser will display the "Tomcat Web Application Manager" as shown in Figure 2.18. If you don't know the correct username or password you may look it up in the file `tomcat-users.xml` as described in Appendix C.2.

After logging successfully into Tomcats manager application, you can select the WAR file(s) to be deployed using button "Choose File" according to the right hand side of Figure 2.18. After picking your `helloworld_server.war` and `helloworld.war` file and closing the file chooser, click on button "Deploy" (located below button "Choose File") to deploy the application to the Tomcat web server. This will copy the selected WAR file into Tomcats `webapps` directory and unpack its content into a subdirectory with the same name. Deploying the file `helloworld.war` will extract its contents into a subdirectory named `helloworld` as shown in Figure **??**. Accordingly, the file `helloworld_server.war` will be extracted into subdirectory `helloworld_server`. You can now connect to the deployed application using the browser of your choice and enter the following address.

```
http://localhost:8080/helloworld_server/
```

You will then see the home page of the server of your "Hello World" application shown in Figure 2.19. From here you can download the zipped client application that can be saved in a directory of your choice. After unpacking the zip file, you may start the executable file named `helloworld`. This will start the "Hello World" client application as shown on the left hand side of Figure 2.20. To start the "Hello World" web application, open a browser and enter the following address.

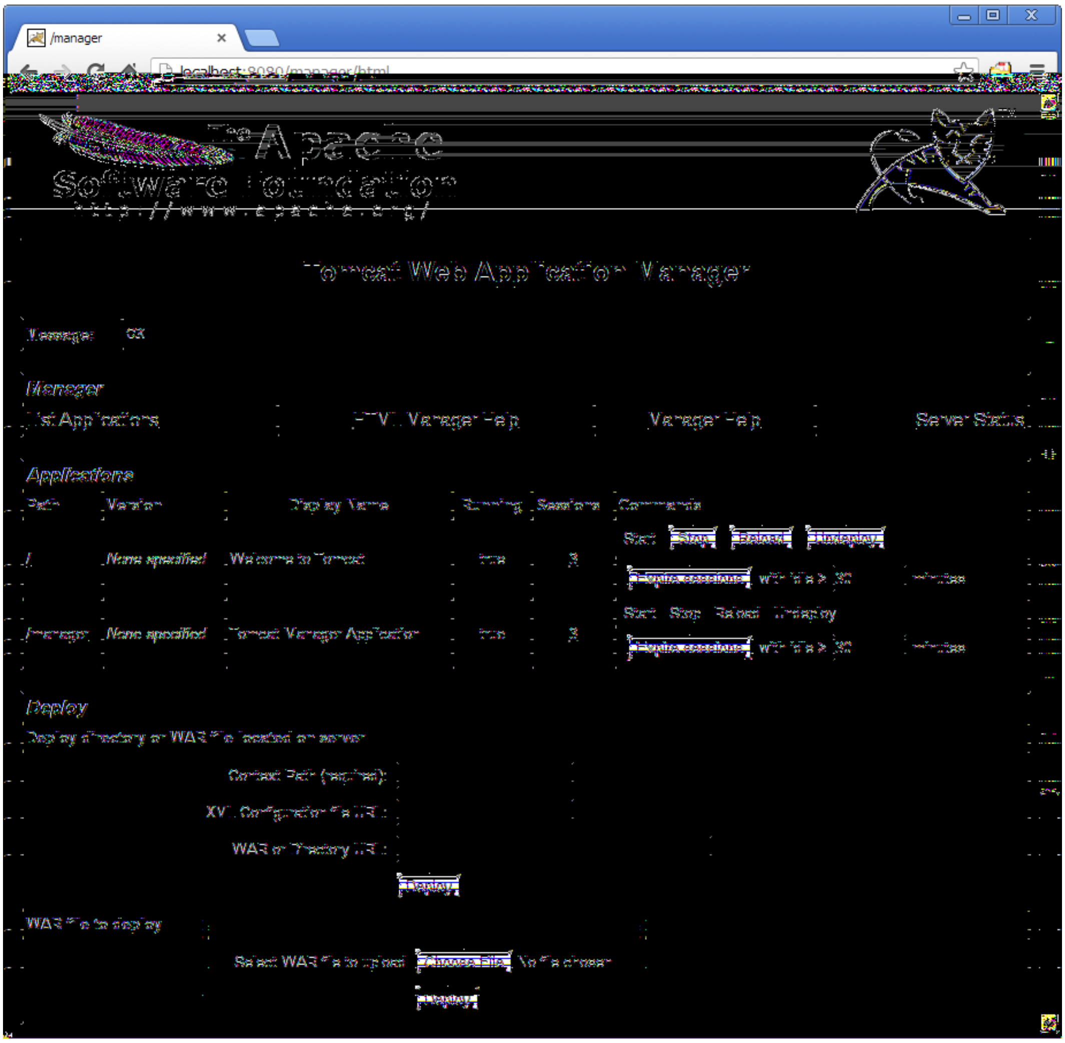
```
http://localhost:8080/helloworld/
```



The server http://localhost:8080 requires a username and password. The server says: Tomcat Manager Application.

Figure 2.17: The Tomcat shown after a successful installation. After clicking on the "Manager App" button (highlighted in red) the login box is shown in front. A successful login shows the "Tomcat Web Application Manager".

Depending on the device your browser is running on you will be redirected to `helloworld/web` on a desktop or laptop computer, to `helloworld/mobile` on a mobile device or to `helloworld/mobile` if you are connecting from a tablet device. Figure 2.20 shows screenshots for a desktop client, the web application and the same application in a mobile browser. As demonstrated in these screenshots `helloworld/web` and `helloworld/mobile` lead to a different presentation of the same UI optimized to the target form factors of desktop browsers, tablets, and mobile phones.



Tomcat Web Application Manager

Message: OK

Manager

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*	Welcome to Tomcat	true	3	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	*None specified*	Tomcat Manager Application	true	3	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

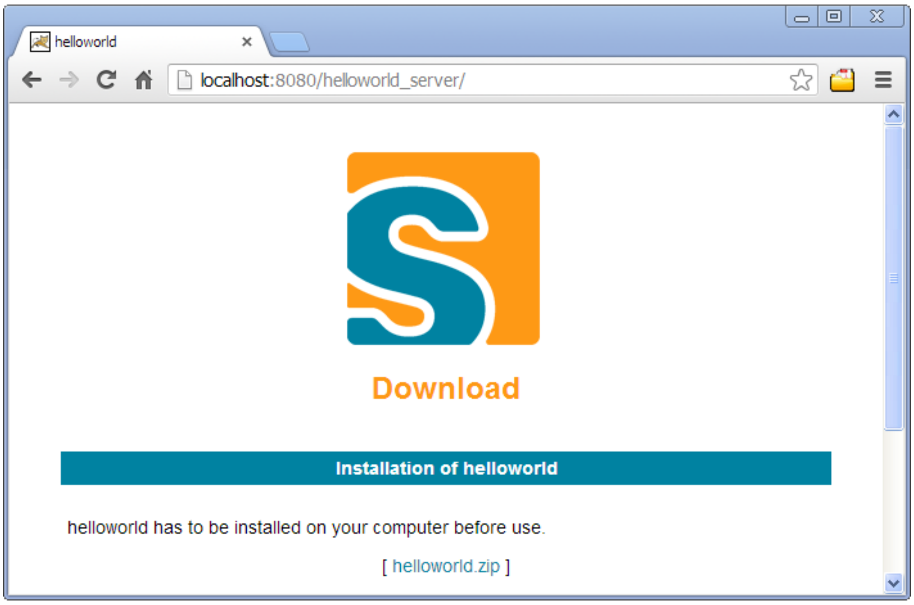
XML Configuration file URL:

WAR or Directory URL:

WAR file to deploy

Select WAR file to upload No file chosen

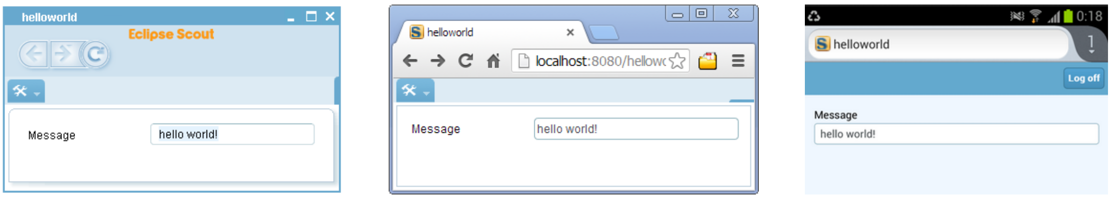
Figure 2.18: The "Tomcat Web Application Manager". The WAR files to be deployed can then be selected using button "Choose File" highlighted in red.



helloworld has to be installed on your computer before use.

[helloworld.zip]

Figure 2.19: The "Hello World" home page, providing a link to download the desktop client.



Message: hello world!

Figure 2.20: The "Hello World" client application running on the desktop, in the browser and on a mobile device.

Chapter 3

"Hello World" Background

The previous "Hello World" tutorial has been designed to cover the creation of a complete client server application in a minimal amount of time. In this chapter, we will take a deeper look at the "Hello World" and provide background information along the way. The goal is to explain many of the used concepts in the context of a concrete Scout application to allow for a well rounded first impression of the Eclipse Scout framework and the tooling provided by the Scout SDK.

The structure of this chapter is closely related to the "Hello World" tutorial. As you will notice, the order of the material presented here exactly follows the previous tutorial and identical section titles are used where applicable. In addition to Chapter 2, we include Section 3.2 to discuss the initial application generated by the Scout SDK.

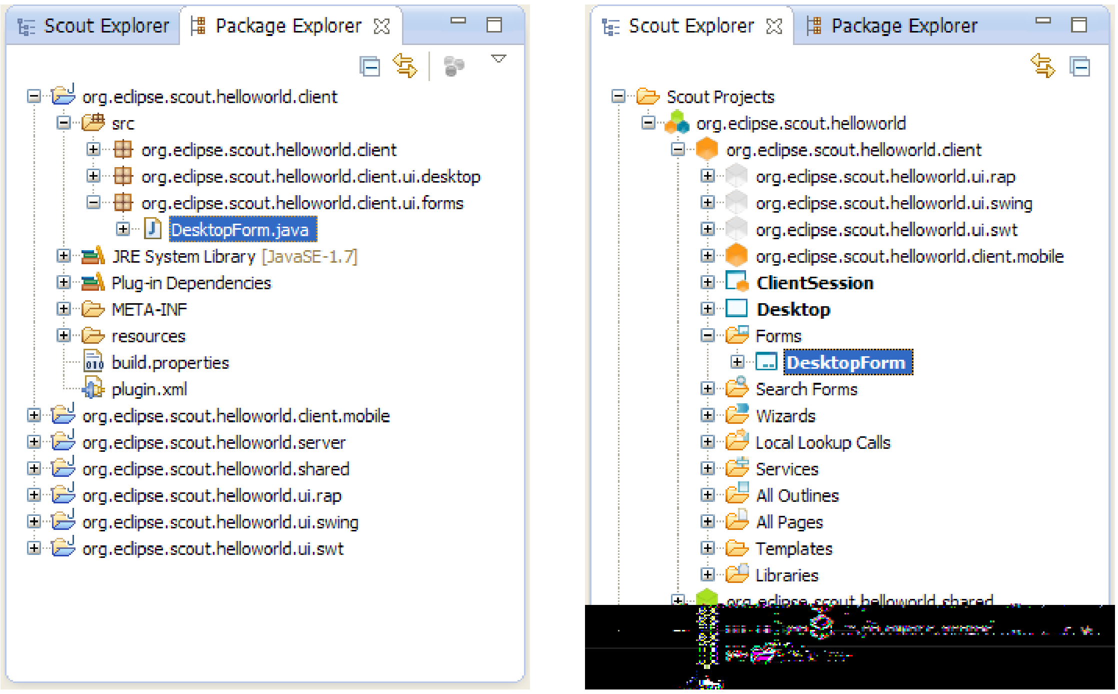
3.1 Create a new Project

The first thing you need for the creation of a new Scout project is to select a new workspace. For Eclipse, a workspace is a directory where Eclipse can store a set of projects in a single place. As Scout projects typically consist of several Eclipse plugin projects the default (and recommended) setting is to use a single workspace for a single Scout project.

In the case of the "Hello World" application, the workspace contains seven plugin projects as shown on the left side of Figure 3.1. In the expanded source folder of the client plugin *org.eclipse.scout.helloworld.client* the organisation of the Java packages is revealed. The Scout Explorer provided on the right side of Figure 3.1 shows three colored top level nodes below the main project *org.eclipse.scout.helloworld.*

In the Scout Explorer, the main project node expands to the orange client node *org.eclipse.scout.helloworld.client*, the green shared node *org.eclipse.scout.helloworld.client* and the blue server node *org.eclipse.scout.helloworld.server*. The client node first presents the white user interface (UI) nodes *org.eclipse.scout.helloworld.client.ui.** indicating the supported UI technologies. Next, the client mobile node *org.eclipse.scout.helloworld.client.mobile* is shown. It is responsible for adapting the layout of the user interface suitably for mobile and tablet devices. Finally, after the *ClientSession* node and the *Desktop* node, component specific folders allow for a simple navigation to the various client parts.

Comparing the Package Explorer with the Scout Explorer a couple of aspects are notable.



- org.eclipse.scout.helloworld.client
  - src
    - org.eclipse.scout.helloworld.client
    - org.eclipse.scout.helloworld.client.ui.desktop
    - org.eclipse.scout.helloworld.client.ui.forms
      - DesktopForm.java
  - JRE System Library [JavaSE-1.7]
  - Plug-in Dependencies
  - META-INF
  - resources
  - build.properties
  - plugin.xml
- org.eclipse.scout.helloworld.client.mobile
- org.eclipse.scout.helloworld.server
- org.eclipse.scout.helloworld.shared
- org.eclipse.scout.helloworld.ui.rap
- org.eclipse.scout.helloworld.ui.swing
- org.eclipse.scout.helloworld.ui.swt

- Scout Projects
  - org.eclipse.scout.helloworld
    - org.eclipse.scout.helloworld.client
      - org.eclipse.scout.helloworld.ui.rap
      - org.eclipse.scout.helloworld.ui.swing
      - org.eclipse.scout.helloworld.ui.swt
      - org.eclipse.scout.helloworld.client.mobile
      - **ClientSession**
      - **Desktop**
      - Forms
        - **DesktopForm**
      - Search Forms
      - Wizards
      - Local Lookup Calls
      - Services
      - All Outlines
      - All Pages
      - Templates
      - Libraries
    - org.eclipse.scout.helloworld.shared

Figure 3.1: The Eclipse plugin projects of the "Hello World" application shown by the Package Explorer in the Scout SDK on the left hand side. The corresponding view in the Scout Explorer is provided on the right hand side.

First, the number and names of the Eclipse plugin projects is identical in both the Package Explorer and the Scout Explorer view. However, the Scout Explorer recognizes the Scout project structure and explicitly renders the relation between the different Eclipse plugins. In addition, individual node colors are used to indicate the role of each plugin project. Second, the focus of the Scout Explorer lies on the business functionality of the complete client server application. Artefacts only necessary to the underlying Eclipse platform are not even accessible. Third, on the individual elements rendered in the Scout Explorer, the Scout SDK provides menus to start wizards useful to the selected context. In the case of the "Hello World" tutorial we could create the complete application (except for a single line of Java code) using these wizards .

When we revisit the *New Scout Project* wizard in Figure 2.2, it now becomes trivial to explain how the *Project Name* field `org.eclipse.scout.helloworld` was used as the common prefix for plugin project names and Java package names. Based on the project name, the last part `helloworld` was used for the *Project Alias* field. As we have seen in Section 2.7, this project alias is used by the Scout SDK to build the base names of the WAR files in the export step. In turn, after deploying the WAR files as described in Section 2.8, the RAP server application becomes available under the URL `http://localhost:8080/helloworld`. Should you have a catchy naming for you application in mind, `com.mycompany.mycatchyname` is therefore a good choice for the *Project Name* field.

3.2 Walking through the Initial Application

In this section, we will walk you through the central Scout application model components of the "Hello World" example. As each of these components is represented by a Java class in the Scout framework, we can explain the basic concept using the available "Hello World" source code. Below, we will introduce the following Scout components.

- Desktop
- Form
- Form handler
- Service
- MainBox
- Form data
- Form field

Please note that most of the Java code was initially generated by Scout SDK. In many cases this code can be used "as is" and does not need to be changed. Depending on your requirements, it might very well be that you want to adapt the provided code to fit your specific needs. However, a basic understanding of the most important Scout components should help you to better understand the structure and working of Scout applications.

Listing 3.1: Creating and starting a form in the client's Desktop callback method `execOpened`.

```
@Override
protected void execOpened() throws ProcessingException {
  //If it is a mobile or tablet device, the DesktopExtension in the mobile ↘
    →plugin takes care of starting the correct forms.
  if (!UserAgentUtility.isDesktopDevice()) {
    return;
  }
  DesktopForm desktopForm = new DesktopForm();
  desktopForm.setIconId(Icons.EclipseScout);
  desktopForm.startView();
}
```

3.2.1 Desktop

The desktop is the central container of all visible elements of the Scout client application. It inherits from Scout class `AbstractDesktop` and represents the empty application frame with attached elements, such as the applications menu tree. In the ”Hello World” application, it is the Desktop that is first opened when the user starts the client application.

To find the desktop class in the Scout Explorer, we first navigate to the orange *client* node and double click the *Desktop* node just below. This will open the associated Java file `Desktop.java` in the editor view of the Scout SDK. Of interest is the overwritten callback method `execOpened` shown in Listing 3.1.

Method `execOpened` is called by the Scout framework after the desktop frame becomes visible. The only thing that happens here is the creation of a `desktopForm` object, that gets assigned an icon before it is started via method `startView`. This desktop form object holds the *Message* field text widget that is displayed to the user[1]. More information regarding form elements are provided in the next section.

3.2.2 Form

Scout forms are UI containers that hold form field widgets. A Scout form always inherits from Scout class `AbstractForm` and can either be displayed as a dialog in its own window or shown as a view inside of another UI container. In the ”Hello World” application a `DesktopForm` object is created and displayed as a view inside of the desktop element.

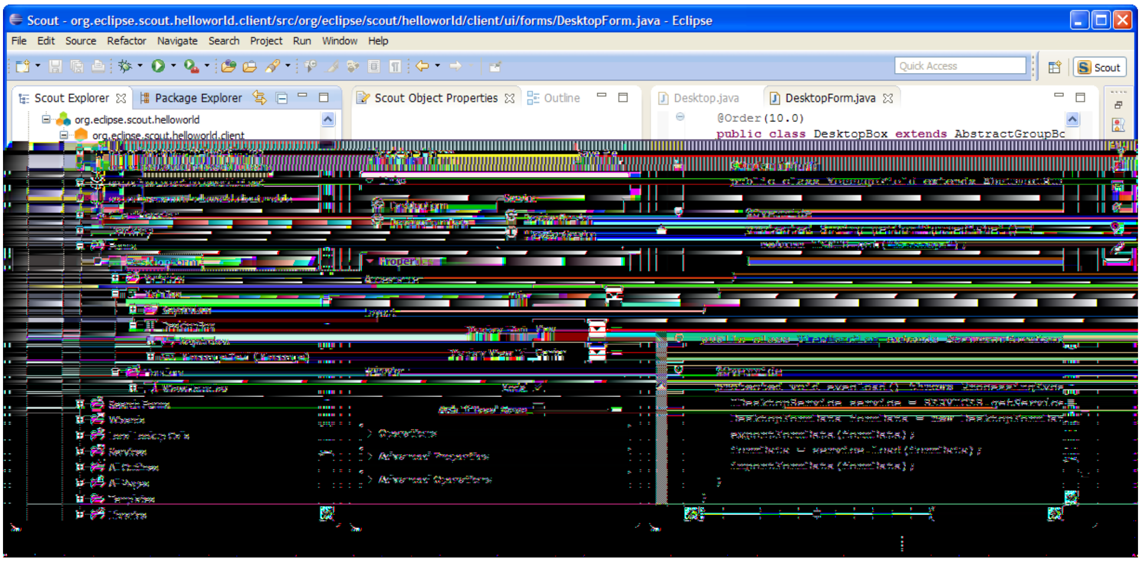
To find the desktop form class in the Scout Explorer, expand the orange *client* node[2]. Below this node, you will find the *Forms* folder. Expand this folder to show the *DesktopForm* as shown in Figure 3.2. In the Scout Object Property window in the screenshot, we can also see the *Display Hint* property. Its value is set to 'View' to display the desktop form as a view and not as a dialog in its own frame.

Expand the *DesktopForm* to show its children: *Variables*, *MainBox* and *Handlers*. The *Variables* sub folder contains variables. They are invisible to the application user. The ”Hello World” application is so simple, it does not need variables. The sub folder *MainBox* contains form fields.

---

[1] In the Scout application model we can only add UI fields to Scout form elements, not directly to the desktop.

[2] To expand elements (nodes, folders, etc.) in the Scout Explorer, use a double click on the element or a single click on the plus icon in front of the element.


Scout - org.eclipse.scout.helloworld.client/src/org/eclipse/scout/helloworld/client/ui/forms/DesktopForm.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Scout
Scout Explorer
Package Explorer
Scout Object Properties
Outline
Desktop.java
DesktopForm.java
org.eclipse.scout.helloworld
org.eclipse.scout.helloworld.client

```
@Order(10.0)
public class DesktopBox extends AbstractGroupBo
```

Figure 3.2: Scout SDK showing the DesktopForm's *ViewHandler* in the Scout Explorer and the form properties in the Scout Object Properties.

These are the visible user interface elements. The main box of our `DesktopForm` holds the `DesktopBox` containing the `MessageField` added with the *New Form Field* wizard. Finally, the *Handlers* sub folder contains all available form handlers. The view handler shown in Figure 3.2 has been added in the initial project creation step.

3.2.3 Form Handler

Form handlers are used to manage the form's life cycle. Scout form handlers inherit from `AbstractFormHandler` and allow the implementation of desired behaviour before a form is opened, or after it is closed. This is achieved by overwriting callback methods defined in `AbstractFormHandler`. The necessary wiring is provided by the Scout framework, either by the initial project creation step or when using one of the provided Scout SDK wizards.

In the "Hello World" application, it is the overwritten `execLoad` method in the `ViewHandler` that defines what will happen before the desktop form is shown to the user. The corresponding source code is provided in Listing 3.2. It is this `execLoad` method where most of the behaviour relevant to the "Hello World" application is implemented. Roughly, this implementation is performing the following steps.

1. Get a reference to the forms server service running on the server.
2. Create a data transfer object (DTO)[3]
3. Pass the empty DTO to the load service method (ask the server for some data).
4. Update the DTO with the content provided by the service load method.
5. Copy the updated information from the DTO into the desired form field.

---

[3] Data Transfer Object (DTO): `http://en.wikipedia.org/wiki/Data_transfer_object`.

Listing 3.2: Class `DesktopForm` with its view handler and `startView` method. Other inner classes and methods are omitted here.

```
public class DesktopForm extends AbstractForm {

 public class ViewHandler extends AbstractFormHandler {

   @Override
   protected void execLoad() throws ProcessingException {
     IDesktopService service = SERVICES.getService(IDesktopService.class);
     DesktopFormData formData = new DesktopFormData();
     exportFormData(formData);
     formData = service.load(formData);
     importFormData(formData);
   }
 }

 public void startView() throws ProcessingException {
   startInternal(new ViewHandler());
 }
}
```

To open the `ViewHandler` class in the Java editor of the Scout SDK, double click on the *ViewHandler* in the Scout Explorer. Your Scout SDK should then be in a state similar to Figure 3.2. In the lower part of Listing 3.2 we can see the wiring between the desktop form and the view handler in method `startView`. Further up, we find method `execLoad` of the view handler class.

Before we discuss this method’s implementation, let us examine when and how `execLoad` is actually called. As we have seen in the `Desktop` class (see Listing 3.1), the form’s method `startView` is executed after the desktop form is created. Inside method `startView` (see Listing 3.2), the desktop form is started/opened using `startInternal`. In method `startInternal` a view handler is then created and passed as a parameter. This eventually leads to the call of our `execLoad` custom implementation.

We are now ready to dive into the implementation of method `execLoad` of the desktop form’s view handler. First, a reference to a form service identified by `IDesktopService` is obtained using `SERVICES.getService`. Then, a form data object (the DTO) is created and all current form field values are exported into the form data via method `exportFormData`. Strictly speaking, the `exportFormData` is not necessary for the use case of the ”Hello World” application. But, as this is generated code, there is no benefit when we manually delete the `exportFormData` command. Next, using the `load` service method highlighted in Listing 3.2, new form field values are obtained from the server and assigned to the form data object. Finally, these new values are imported from the form data into the form via the `importFormData` method. Once the desktop form is ready, showing it to the user is handled by the framework.

To add some background to the implementation of the `execLoad` above, the next section introduces services and form data objects.

3.2.4 Form Services and Form Data Objects

Form services and form data objects are used in the Scout framework to exchange information between the Scout client and server applications. When needed, a service implemented on the server side can register a corresponding proxy service on the client. This proxy service is invoked by the client as if it were implemented locally. In fact, when we get a service reference using `SERVICES.getService`, we do not need to know if this service runs locally on the client or remotely on the server.

In the "Hello World" example application, the client's desktop form has an associated desktop service running on the server. This correspondence between forms and form services is also reflected in the *Links* section of the Scout Object Properties of the desktop form. As shown in Figure 3.2, links are provided not only for the desktop form, but for its desktop form data, the corresponding desktop form service as well as for the service interface `IDesktopService`. On the client, this interface is used to identify and register the proxy service for the desktop service.

To transfer data between the client and the server, the "Hello World" application uses a `DesktopFormData` object as a DTO. This form data object holds all form variables and values for all the form fields contained in the form. Taking advantage of this correspondence, the Scout framework provides the convenience methods `exportFormData` and `importFormData`. As a result, the developer does not need to deal with any mapping code between the form data object and the form fields.

The actual implementation of the desktop form service in class `DesktopService` is implemented on the server side. As the class `DesktopService` represents an ordinary Scout service it inherits from `AbstractService`. It also implements its corresponding `IDesktopService` interface used for registering both the actual service as well as the proxy service.

3.3 Run the Initial Application

3.3.1 The Launcher Boxes

To run a Scout application the Scout SDK provides launcher boxes in the Scout Object Properties as described in Section 2.3. These object properties are associated to the top level project node in the Scout Explorer. Using the EDIT icon provided in the product launcher section of the Scout Object Properties, the list of launcher boxes can be specified as shown in Figure 3.3.

3.3.2 Eclipse Product Files

The available products shown on the right side of Figure 3.3 represent the Eclipse product files created in the initial project creation step. Product files[4] are used in Eclipse to specify the configuration and content of an executable application. In the case of the "Hello World" project, four executable applications — with two Eclipse product files for each application — have been defined by the Scout SDK. The four applications, one for the server application and one for each client technology, have already been discussed in Section 2.3.

The reason why each executable has two product files associated with it lies in the assumption

---

[4] Read the following article for an introduction to Eclipse product files: http://www.vogella.com/articles/EclipseProductDeployment/article.html.

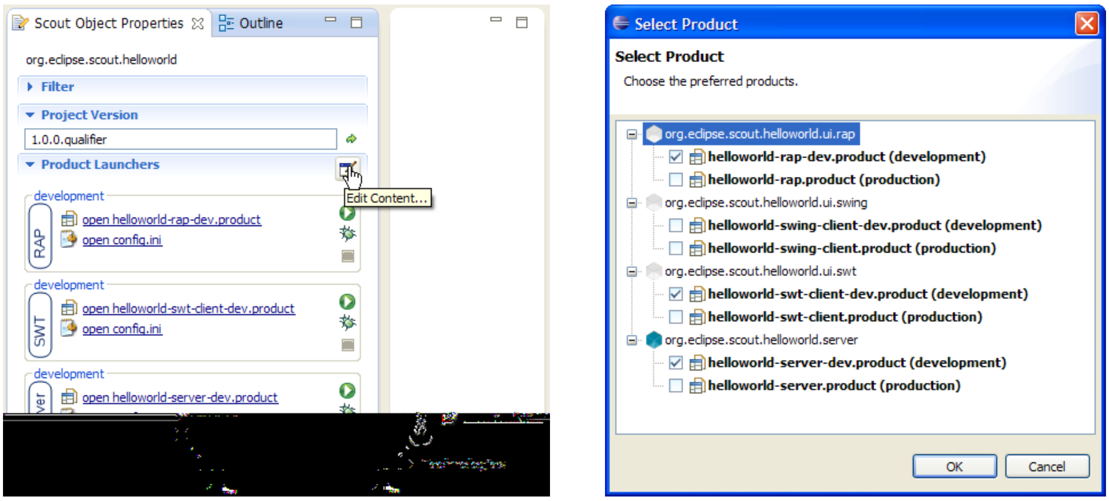

Scout Object Properties
Outline
org.eclipse.scout.helloworld
Filter
Project Version
1.0.0.qualifier
Product Launchers
Edit Content...
development
RAP
open helloworld-rap-dev.product
open config.ini
development
SWT
open helloworld-swt-client-dev.product
open config.ini
development
open helloworld-server-dev.product
Select Product
Select Product
Choose the preferred products.
org.eclipse.scout.helloworld.ui.rap
☑ helloworld-rap-dev.product (development)
☐ helloworld-rap.product (production)
org.eclipse.scout.helloworld.ui.swing
☐ helloworld-swing-client-dev.product (development)
☐ helloworld-swing-client.product (production)
org.eclipse.scout.helloworld.ui.swt
☑ helloworld-swt-client-dev.product (development)
☐ helloworld-swt-client.product (production)
org.eclipse.scout.helloworld.server
☑ helloworld-server-dev.product (development)
☐ helloworld-server.product (production)
OK
Cancel

Figure 3.3: Using the EDIT CONTENT... icon shown on the left hand side, the product selection dialog shown on the right side is opened. Using this product selection dialog, the list of launcher boxes can be specified.

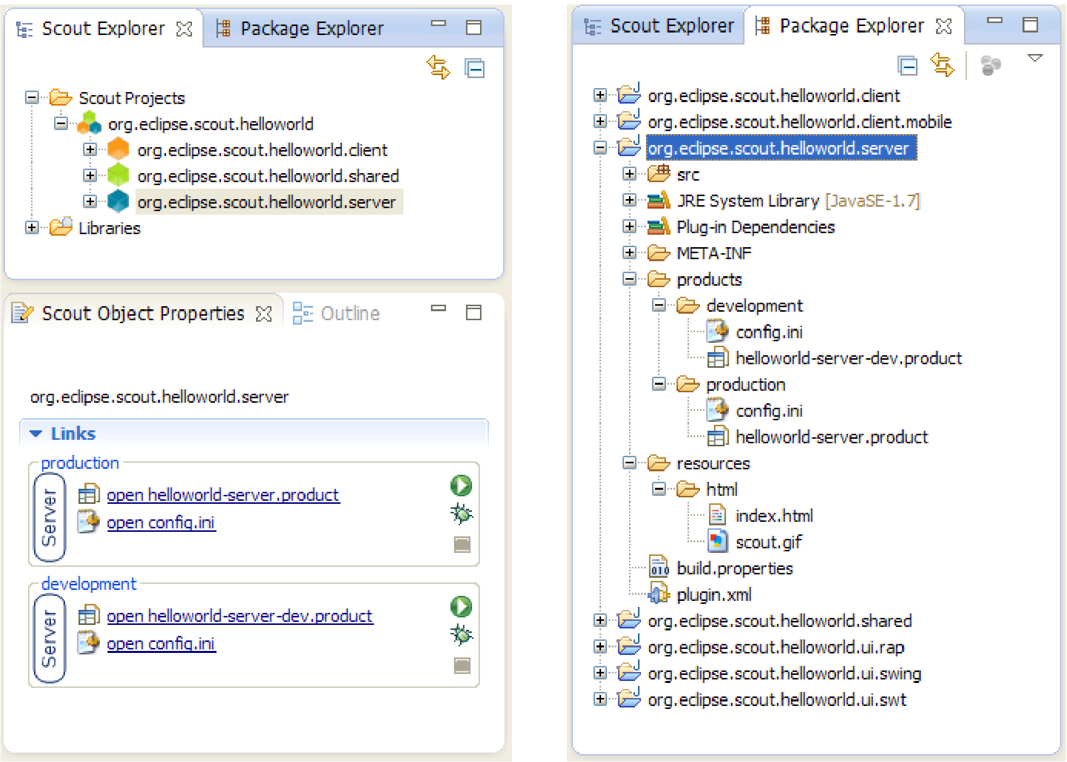
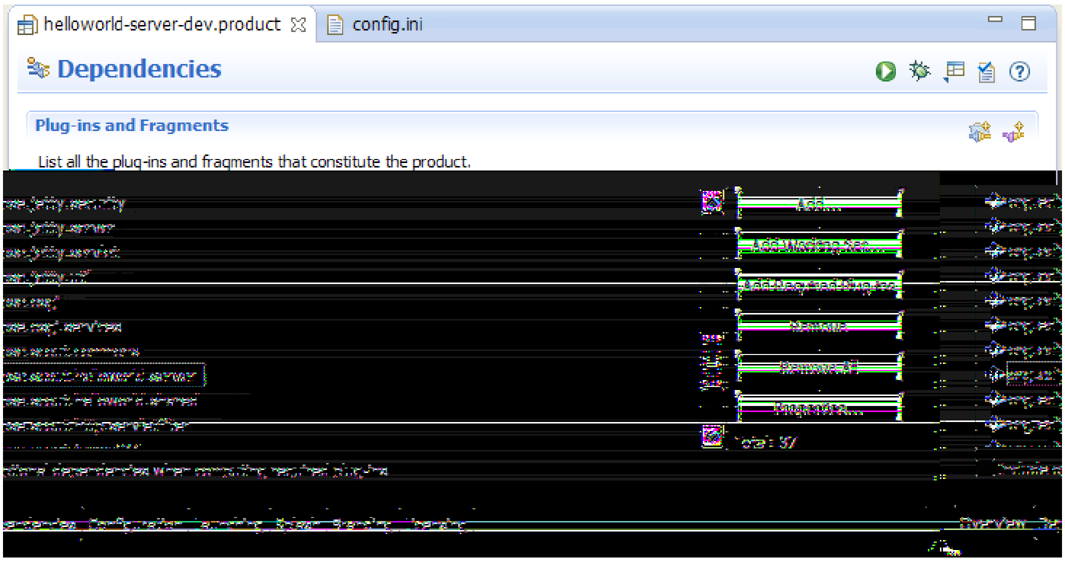

Scout Explorer
Package Explorer
Scout Projects
org.eclipse.scout.helloworld
org.eclipse.scout.helloworld.client
org.eclipse.scout.helloworld.shared
org.eclipse.scout.helloworld.server
Libraries
Scout Object Properties
Outline
org.eclipse.scout.helloworld.server
Links
production
Server
open helloworld-server.product
open config.ini
development
Server
open helloworld-server-dev.product
open config.ini
Scout Explorer
Package Explorer
org.eclipse.scout.helloworld.client
org.eclipse.scout.helloworld.client.mobile
org.eclipse.scout.helloworld.server
src
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
META-INF
products
development
config.ini
helloworld-server-dev.product
production
config.ini
helloworld-server.product
resources
html
index.html
scout.gif
build.properties
plugin.xml
org.eclipse.scout.helloworld.shared
org.eclipse.scout.helloworld.ui.rap
org.eclipse.scout.helloworld.ui.swing
org.eclipse.scout.helloworld.ui.swt

Figure 3.4: The production and development launcher boxes associated with the "Hello World" server application are shown on the left side. In the Package Explorer shown on the right side, the production and development products are located under the *products* folder in the server plugin project.



List all the plug-ins and fragments that constitute the product.

Figure 3.5: The Eclipse product file editor showing file `helloworld-server-dev.product` of the "Hello World" application. In the Dependencies tab shown above, the list of Eclipse plugins that are required for the server application are shown.

that we want to run the Scout application in different environments. Even in the case of the simple "Hello World" example, the Scout application is started in two target environments. The development environment defines the product in the context of the Scout SDK. To export and run the Scout application outside of the Scout SDK, the production product files are used to define the application when it is to be started on a Tomcat web server. Figure 3.4 illustrates this situation for the "Hello World" server application. On the left side, the blue server node is selected in the Scout Explorer. This opens the two server launcher boxes for the production and the development environment. On the right side of Figure 3.4, the corresponding plugin project *org.eclipse.scout.helloworld.server* is expanded to show the file based organisation of the two product definitions.

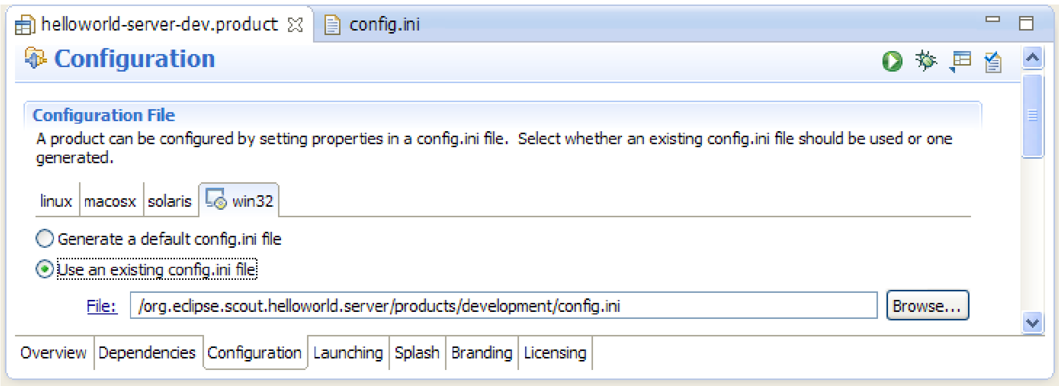
For the case of the "Hello World" example we did not need to edit or change the product files generated by the Scout SDK. However, if your requirements are not met by the provided product files, you may use the Eclipse product file editor. A screenshot of this editor is shown in Figure 3.5 with the tab *Dependencies* opened. In the tab Dependencies, the complete list of necessary plugins is provided. Example plugins visible in Figure 3.5 include the "Hello World" server and shared plugins, Scout framework plugins, and Jetty plugins. The Jetty[5] plugins are only needed to run the "Hello World" server application inside the Scout SDK. Consequently, Jetty plugins are not listed as a dependency in the Scout server's production product file.

3.3.3 Eclipse Configuration Files

Switching to tab *Configuration* in the product file editor, shows the selected radio button *Use an existing config.ini file* and the link to the configuration file provided in the *File* field as shown in the upper part of Figure 3.6. Below, a part of the server's `config.ini` file is shown. Both the

---

[5] Jetty is web server with a small footprint: http://www.eclipse.org/jetty/.



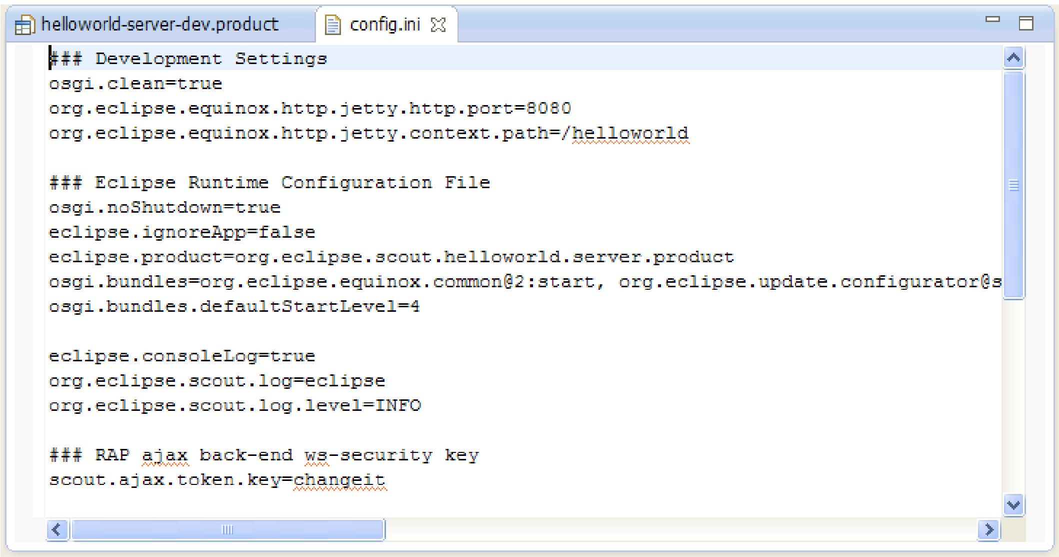
Configuration File

A product can be configured by setting properties in a config.ini file. Select whether an existing config.ini file should be used or one generated.

○ Generate a default config.ini file

● Use an existing config.ini file

File: /org.eclipse.scout.helloworld.server/products/development/config.ini



```
### Development Settings
osgi.clean=true
org.eclipse.equinox.http.jetty.http.port=8080
org.eclipse.equinox.http.jetty.context.path=/helloworld

### Eclipse Runtime Configuration File
osgi.noShutdown=true
eclipse.ignoreApp=false
eclipse.product=org.eclipse.scout.helloworld.server.product
osgi.bundles=org.eclipse.equinox.common@2:start, org.eclipse.update.configurator@s
osgi.bundles.defaultStartLevel=4

eclipse.consoleLog=true
org.eclipse.scout.log=eclipse
org.eclipse.scout.log.level=INFO

### RAP ajax back-end ws-security key
scout.ajax.token.key=changeit
```

Figure 3.6: Above, the definition of the products `config.ini` in tab *Configuration* of the product file editor. Below, the content of the configuration file of the "Hello World" server application is provided in a normal text editor.

entry in the product file pointing to the configuration file, and the content of the `config.ini` file has been generated by the Scout SDK during the initial project creation step. As shown in the lower part of Figure 3.6, Eclipse configuration files have the format of a standard property file. The provided key value pairs are read at startup time if the `config.ini` file can be found in folder `configuration` by the Eclipse runtime.

3.3.4 Scout Desktop Client Applications

Having introduced Eclipse product files and configuration files based on the "Hello World" server application, we will now look at the different client applications in turn. With Swing applications[6] and SWT applications[7], two alternative UI technologies are currently available to build Scout desktop client applications. More recently, JavaFX[8] is promoted as a successor to Swing and it is therefore likely, that Scout will provide JavaFX client applications in the future.

When we compare the product files for the Swing and the SWT client applications, it is apparent that both client applications share a large number of plugins. Most importantly, the complete UI model and the business logic is identical for both client applications. In other words, the value created by the Scout developer is contained in the two plugins *org.eclipse.scout.helloworld.client* and *org.eclipse.scout.helloworld.shared*. To create an executable client application, we only need to combine these two plugins with a set of plugins specific to the desired UI technology.

After starting the "Hello World" Swing client or the corresponding SWT client application, the client application first reads the startup parameters from its `config.ini` file. Among other things, this client configuration file contains the parameter `server.url` to specify the URL to the "Hello World" server. After the startup of the "Hello World" client application, it can then connect to the "Hello World" server application using this address.

3.3.5 Scout Web, Tablet and Mobile Clients

For Scout web, tablet and mobile clients, the Eclipse RAP framework[9] is used. The RAP framework provides an API that is almost identical to the one provided by SWT and allows to use Java for server-side Ajax[10]. This setup implies that Scout tablet and mobile clients are not native clients but browser based[11].

Comparing the product file of the SWT client applications with the RAP application, we observe that the RAP development product does not include any SWT plugins, but a set of RAP and Jetty plugins. In addition, the RAP product also contains the Scout mobile client plugins *org.eclipse.scout.rt.client.mobile* and *org.eclipse.scout.helloworld.client.mobile*. These two plugins are responsible for transforming the UI model defined in the "Hello World" client plugin to the different form factors of tablet computers and mobile phones.

If you start the "Hello World" RAP application in your Scout SDK, you are launching a second server application in a Jetty instance on a different port than the "Hello World" server application.

---

[6] Swing is the primary Java UI technology: `http://en.wikipedia.org/wiki/Swing_%28Java%29`.

[7] Standard Widget Toolkit (SWT): `http://en.wikipedia.org/wiki/Standard_Widget_Toolkit`.

[8] JavaFX is the most recent Java UI technology: `http://en.wikipedia.org/wiki/JavaFX`.

[9] Remote Application Platform (RAP): `http://www.eclipse.org/rap/`.

[10] Asynchronous JavaScript and XML (AJAX): `http://en.wikipedia.org/wiki/Ajax_%28programming%29`.

[11] To provide native clients with Scout, the simplest (commercial) option is most likely Tabris: `http://developer.eclipsesource.com/tabris/`

Listing 3.3: The `DesktopForm` with its inner class `MainBox` containing the desktop box and message field

```java
@FormData(value = DesktopFormData.class, sdkCommand = SdkCommand.CREATE)
public class DesktopForm extends AbstractForm {

  public DesktopForm() throws ProcessingException {
  @Order(10.0)
  public class MainBox extends AbstractGroupBox {

    @Order(10.0)
    public class DesktopBox extends AbstractGroupBox {

      @Order(10.0)
      public class MessageField extends AbstractStringField {

        @Override
        protected String getConfiguredLabel() {
          return TEXTS.get("Message");
        }
      }
    }
  }
}
```

As in the case of the desktop client applications, the RAP or Ajax server application knows how to connect to the "Hello World" server application after reading the parameter `server.url` from its `config.ini` file.

3.4 The User Interface Part

Using the UI of the "Hello World" application we explain in this section how the Scout UI form model is represented in Java. We also describe how this representation is exploited by the Scout SDK to automatically manage the form data objects used for data transfer between Scout client and Scout server applications. Finally, will have a brief look at internationalization[12] support of Scout for texts.

As discussed in Section 3.2.2 Scout forms consist of variables, the main box and a number of form handlers. The main box represents the visible part of Scout's form model. It may holds any number of form fields. Using container fields such as group boxes, it is possible to define complex structures such as hierarchical UI models containing multiple levels. In the Scout framework the forms structure is represented in the form of inner classes that are located inside of the `MainBox` class. And the *New Form Field* wizard of the Scout SDK fully supports this pattern. Listing 3.3 provides the concrete example using the the desktop form of the "Hello World" tutorial.

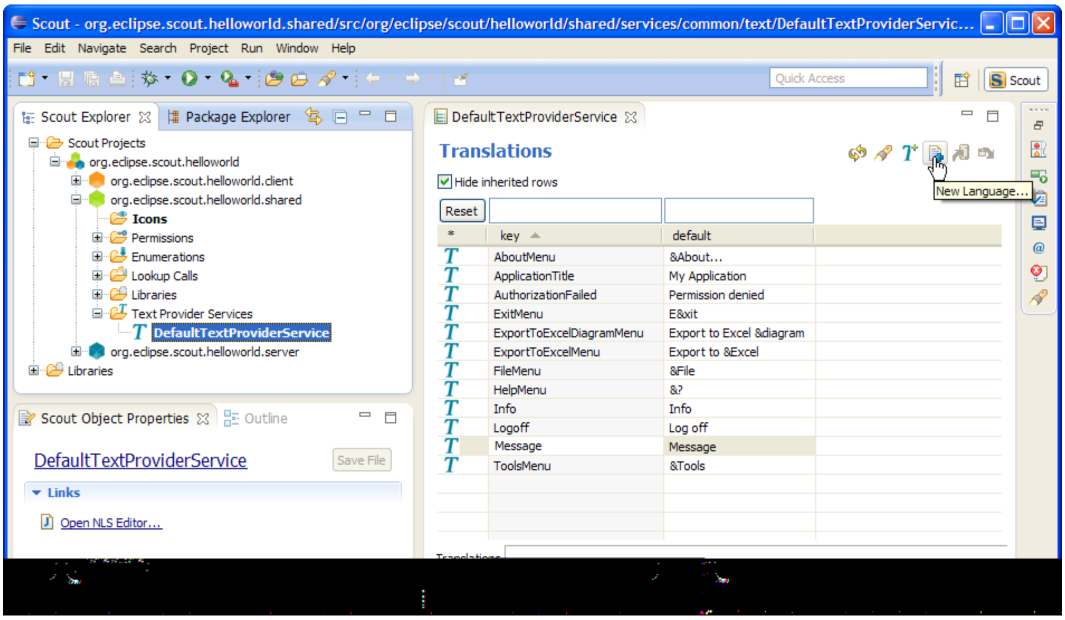
Using inner Java classes to model a form's content is a central aspect of the UI part of the Scout application model. It allows the Scout SDK to easily parse the form's Java code on the fly and directly reflect changes to the form model in the Scout Explorer and the Scout Property

---

[12] Internationalization and localization, also called NLS support: `http://en.wikipedia.org/wiki/Internationalization_and_localization`.

Listing 3.4: The `DefaultTextProviderService` class. Its getter method provides the path and the base name for the text property files

```
public class DefaultTextProviderService extends ↘
 →AbstractDynamicNlsTextProviderService {
 @Override
 protected String getDynamicNlsBaseName() {
   return "resources.texts.Texts";
 }
}
```


Scout - org.eclipse.scout.helloworld.shared/src/org/eclipse/scout/helloworld/shared/services/common/text/DefaultTextProviderServic...
File Edit Navigate Search Project Run Window Help
Quick Access
Scout
Scout Explorer
Package Explorer
Scout Projects
org.eclipse.scout.helloworld
org.eclipse.scout.helloworld.client
org.eclipse.scout.helloworld.shared
Icons
Permissions
Enumerations
Lookup Calls
Libraries
Text Provider Services
DefaultTextProviderService
org.eclipse.scout.helloworld.server
Libraries
Scout Object Properties
Outline
DefaultTextProviderService
Save File
Links
Open NLS Editor...
DefaultTextProviderService
Translations
New Language...
Reset

☑ Hide inherited rows

*	key	default
T	AboutMenu	&About...
T	ApplicationTitle	My Application
T	AuthorizationFailed	Permission denied
T	ExitMenu	E&xit
T	ExportToExcelDiagramMenu	Export to Excel &diagram
T	ExportToExcelMenu	Export to &Excel
T	FileMenu	&File
T	HelpMenu	&?
T	Info	Info
T	Logoff	Log off
T	Message	Message
T	ToolsMenu	&Tools

Figure 3.7: The NLS editor provided by the Scout SDK. This editor is opened via the *Open NLS Editor ...* link in the Scout Object Properties of the *DefaultTextProviderService* node.

View. However, this is not the only benefit for the Scout SDK. As form data objects hold all form variables and the values of all form fields contained in the form, the Scout SDK can keep the form data classes in sync with the forms of the application. It is important to note that this mechanism only depends on the Java code of the form field class. In consequence, the Scout SDK can update form field classes in the background even when form fields are manually coded into the form's Java class. This includes adding all the necessary getter and setter methods to access the values of all the fields defined on a form. As a result, Scout developers don't need to manually update form data objects when the UI model of a form is changed. The Scout SDK takes care of this time consuming and error prone task.

When we did add the *Message* field to the desktop form of the "Hello World" application we had to enter a new translation entry for the label of the message field as shown in Figure 2.8. The individual translation entries are then stored in language specific text property files. To modify translated texts we can use the NLS editor[13] provided by the Scout SDK as shown in Figure 4.16.

---

[13] See Section 4.5.4 for a detailed description of the NLS editor.

Listing 3.5: The server service class `DesktopService`.

```
public class DesktopService extends AbstractService implements ↘
  →IDesktopService {

  @Override
  public DesktopFormData load(DesktopFormData formData) throws ↘
    →ProcessingException {
    formData.getMessage().setValue("hello world!");
    return formData;
  }
}
```

To access the translated label field entry in the application, the Scout SDK generated the implementation of `getConfiguredLabel` using `TEXTS.get("Message")` as shown in Listing 3.3. In the default Scout project setup, calling `TEXTS.get` uses the `DefaultTextProviderService` in the background. This text provider service then defines the access path for the text property files to use for the translation. To resolve the provided key, the user’s locale settings are used to access the correct text property file.

3.5 The Server Part

In this background section we take a closer look at Scout services and calling service methods remotely. We will first discuss the setup of an ordinary Scout service. Then, the additional components to call service methods remotely are considered. To explain the concepts in a concrete context, we use the setup of the `DesktopService` of our ”Hello World” example.

3.5.1 Scout Services

Scout services are OSGi services[14] which in turn are defined by standard Java classes or interfaces. Scout is just adding a convenience layer to cover typical requirements in the context of client server applications. To support Scout developers as much as possible, the Scout SDK offers wizards that generate the necessary classes and interfaces and also take care of service registration.

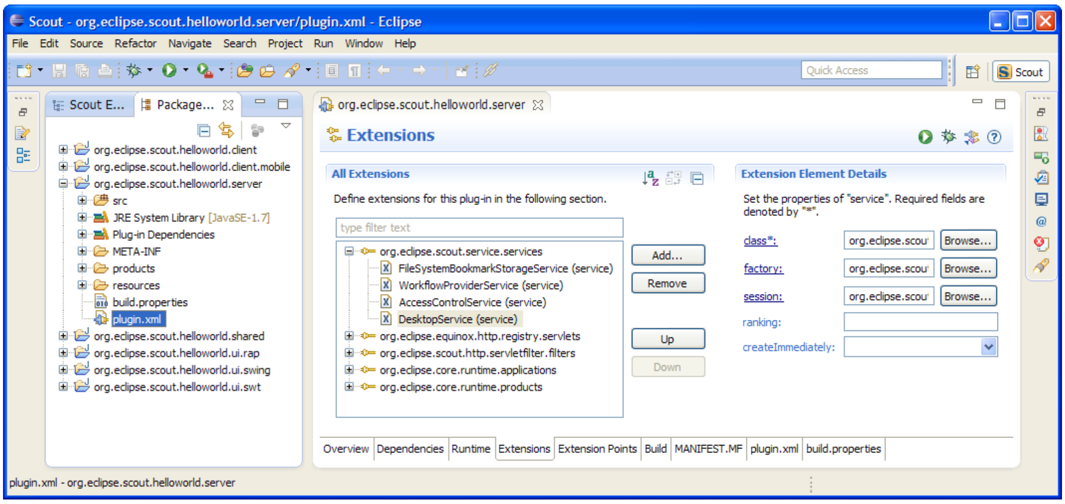
All Scout services need to extend Scout’s `AbstractService` class and implement their own corresponding interface. This also applies to the ”Hello World” desktop service according to Listing 3.5. As shown in Figure 2.10, this service can be located in the Scout Explorer under the blue server node in the *Services* folder.

Before Scout services can be accessed and used, they need to be explicitly registered as a service in the correct place. For this registration mechanism, Scout is using Eclipse extension points and extensions[15] which are conceptually similar to electrical outlets and plugs. And in order to work, as in the case of outlets and plugs, the plug must fit to the outlet. In our ”Hello World” example, the extension (plug) is represented by class `DesktopService` and the service

---

[14] A good introduction to OSGi services is provided by Lars Vogel’s tutorial: `http://www.vogella.com/articles/OSGiServices/article.html`.

[15] A good introduction to Eclipse extensions and extension points is provided in the Eclipse wiki: `http://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points%3F`.


Scout - org.eclipse.scout.helloworld.server/plugin.xml - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Scout
Scout E...
Package...
org.eclipse.scout.helloworld.client
org.eclipse.scout.helloworld.client.mobile
org.eclipse.scout.helloworld.server
src
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
META-INF
products
resources
build.properties
plugin.xml
org.eclipse.scout.helloworld.shared
org.eclipse.scout.helloworld.ui.rap
org.eclipse.scout.helloworld.ui.swing
org.eclipse.scout.helloworld.ui.swt
org.eclipse.scout.helloworld.server
Extensions
All Extensions
type filter text
org.eclipse.scout.service.services
FileSystemBookmarkStorageService (service)
WorkflowProviderService (service)
AccessControlService (service)
DesktopService (service)
org.eclipse.equinox.http.registry.servlets
org.eclipse.scout.http.servletfilter.filters
org.eclipse.core.runtime.applications
org.eclipse.core.runtime.products
Add...
Remove
Up
Down
Extension Element Details
Browse...
Browse...
Browse...
Overview
Dependencies
Runtime
Extensions
Extension Points
Build
MANIFEST.MF
plugin.xml
build.properties
plugin.xml - org.eclipse.scout.helloworld.server

Define extensions for this plug-in in the following section.

Set the properties of "service". Required fields are denoted by "*".

class*: org.eclipse.scou
factory: org.eclipse.scou
session: org.eclipse.scou
ranking:
createImmediately:

Figure 3.8: The Eclipse plugin editor for `plugin.xml` files. In the tab *Extensions* the "Hello World" desktop service is registered under the extension point `org.eclipse.scout.service.services`.

extension point (outlet) is named `org.eclipse.scout.service.services`. What makes the desktop service fit to the service extension point is the fact that its interface `IDesktopService` extends Scout's `IService` interface.

The registration of the desktop service under the service extension point is then defined in the `plugin.xml` file of the "Hello World" server plugin. As shown in Figure 3.8, the `plugin.xml` file is located in the root path of plugin `org.eclipse.scout.helloworld.server`. To modify a `plugin.xml`, you can either use the Eclipse plugin editor or your favorite text editor. In Figure 3.8, the registration of the desktop service is shown in the *Extensions* tab of the plugin editor. For the corresponding XML representation in the `plugin.xml` file, see Listing 3.6.

3.5.2 Scout Proxy Services

In the "Hello World" application the `load` method of the desktop service is called remotely from the client. But so far, we have only seen how the desktop service is implemented and registered in the server application. To call server service methods remotely from Scout client applications, the Scout framework provides client proxy services and the service tunnel. As the name implies, a client proxy service acts as a local proxy service (running in the Scout client application) of a server service (running remotely in the Scout server application).

Client proxy services are defined by a Java interface located in the shared plugin of the Scout application. As shown in Listing 3.5 of the desktop service, this service interface is also implemented by the desktop service class in the server plugin. Corresponding to the registration of the desktop service in the server plugin, client proxy services need to be registered in the client's `plugin.xml` file. The content of the "Hello World" client plugin configuration file is provided in Listing 3.7. To create proxy services in Scout clients, the `ClientProxyServiceFactory` is used. This is also reflected in the extension defined in Listing 3.7. Internally, this service factory then uses the service tunnel to create the local proxy services.

To call a remote service method from the Scout client application, we first need to obtain

Listing 3.6: The registration of the `DesktopService` in the server's `plugin.xml` configuration file. The remaining content of the file has been omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>

  <extension
      name=""
      point="org.eclipse.scout.service.services">
    <service
        factory="org.eclipse.scout.rt.server.services.ServerServiceFactory"
        class="org.eclipse.scout.helloworld.server.services.DesktopService"
        session="org.eclipse.scout.helloworld.server.ServerSession">
    </service>
  </extension>

</plugin>
```

Listing 3.7: The registration of the `IDesktopService` proxy service in the client plugin of the "Hello World" application. This is the complete content of the client's `plugin.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>

  <extension
      name=""
      point="org.eclipse.scout.service.services">
    <proxy
        factory="org.eclipse.scout.rt.client.services.↘
          →ClientProxyServiceFactory"
        class="org.eclipse.scout.helloworld.shared.services.IDesktopService"↘
          →>
    </proxy>
  </extension>

</plugin>
```

a reference to the proxy service. Using the `SERVICES.getService` method with the interface `IDesktopService`, we can obtain such a reference as shown in Listing 3.2 for the view handler of the desktop form. With this reference to the client's proxy service, calling methods remotely works as if the service would be running locally. Connecting to the server, serializing the method call including parameters (and de serializing the return value) is handled transparently by Scout.

3.6 Add the Rayo Look and Feel

Rayo has been designed in 2009 by BSI for its CRM[16] application and contact center solution. Since then, Rayo has been copied for Scout web applications and also adapted to work on touch/mobile devices.

The implementation of Rayo for desktop clients is based on the Java Synth look and feel[17]. However, in a few cases it was necessary to adjust some of the synth classes. In order to do this, the adapted classes are copied form the OpenJDK implementation[18] As OpenJDK is licenced under the GNU General Public Licence (GPL) with a linking exception it is not possible to distribute Rayo under the Eclipse Public Licence. That is why Rayo is not initially contained in the Eclipse Scout package but needs to be downloaded from the Eclipse Marketplace. Fortunately, there is still no restriction to use Rayo in commercial products. The only remaining restriction applies to modifying Rayo for commercial products. In this case you will be obliged to redistribute your modified version of Rayo under the same licence (GPL with classpath exception).

With Eclipse Scout 3.8 (Juno), the Scout framework also allows to build web clients based on Eclipse RAP. Great care has been taken to ensure, that the look and feel for Scout web applications matches the look and feel of the desktop as closely as possible. As RAP is already distributed under the EPL licence the Rayo for web apps is directly contained in the Scout package. TODO: Describe what to change to use RAP default look and feel

A similar approach was chosen for Rayo on tablets and mobile devices that are supported with Eclipse Scout 3.9 (Kepler). For such devices optimized components are used to take into account the smaller screens and the absence of a mouse (no context menus!) But as far as possible, the Rayo look and feel also applies to touch devices. TODO: Pointer to more info regarding mobile devices.

3.7 Exporting the Application

In this background section we look at the content and organisation of the two WAR files generated by the Scout SDK *Export Scout Project* wizard. The first WAR file holds the Scout server including a landing page to download the Scout desktop client. The desktop client is provided in the form of a standalone ZIP file. In the second WAR file, the Ajax server based on Eclipse RAP is contained. This Ajax server provides the URLs that can be accessed by web browsers running on desktop computers or tablet and mobile devices.

The content and its organisation of the exported WAR files was not specifically designed for

---

[16] Customer Relationship Management (CRM): https://en.wikipedia.org/wiki/Customer_relationship_management

[17] Java Synth Look and Feel: http://en.wikipedia.org/wiki/Synth_Look_and_Feel

[18] OpenJDK is an open source implementation of the Java platform: http://openjdk.java.net/.

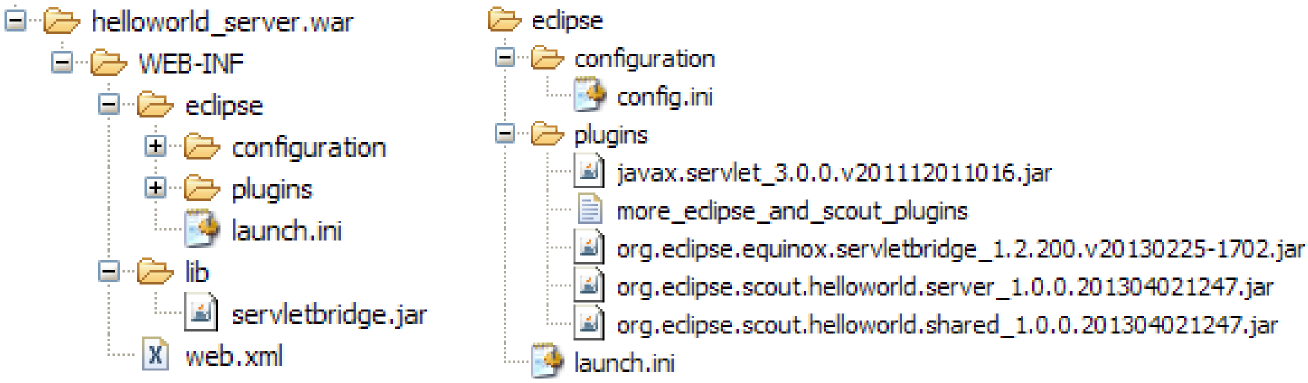

helloworld_server.war
WEB-INF
eclipse
configuration
plugins
launch.ini
lib
servletbridge.jar
web.xml
eclipse
configuration
config.ini
plugins
javax.servlet_3.0.0.v201112011016.jar
more_eclipse_and_scout_plugins
org.eclipse.equinox.servletbridge_1.2.200.v20130225-1702.jar
org.eclipse.scout.helloworld.server_1.0.0.201304021247.jar
org.eclipse.scout.helloworld.shared_1.0.0.201304021247.jar
launch.ini

Figure 3.9: The organisation of the ”Hello World” server WAR file. The right side reveals the location of the `config.ini` file and the application’s plugin files

Scout applications. Rather, it is defined according to server-side Equinox[19], the typical setup for running Eclipse based server applications on a web server. Using file `helloworld_server.war` as a concrete example, we will first describe the general organisation of the WAR file. Then, we introduce individual artefacts of interest that are contained in this WAR file.

The explicit organisation of the server WAR file is shown in Figure 3.9. From the left hand side of the figure we can see that on the top level only folder `WEB-INF` exists in the WAR file. This folder contains all files and directories that are private to the web application. Inside, the web deployment descriptor file `web.xml` as well as the directories `lib` and `eclipse` are located. While the `web.xml` file and directory `lib` are standard for servlet based based applications[20], directory `eclipse` contains all necessary artefacts for servlet based Eclipse applications[21]. Such as Eclipse Scout server applications.

On the right hand side of Figure 3.9 the eclipse specific content of the WAR file is shown. From top to bottom we find the configuration file `config.ini` introduced in Section 3.3.3. In folder `plugins` the necessary plugins that constitute the eclipse application are located where the plugins are available in the form of JAR files[22]. This includes plugins for servlet management, the eclipse platform including the servlet bridge, the scout framework parts and of course our ”Hello World” server and shared plugin. These ”Hello World” jar files exactly match with the plugin projects discussed in Section 3.1.

In Figure 3.10 some of the content of the ”Hello World” server plugin is shown. The first thing to note is that the plugin file conforms to the JAR file format including a `META-INF/MANIFEST.MF` file and the directory tree containing the Java class files, as the `DesktopService.class` implemented in Section 2.5 of the ”Hello World” tutorial. In folder `resources/html` the necessary files for the download page shown in Figure 2.19 including the zipped desktop client are contained. To access this download page the Scout server’s resource servlet `ResourceServlet` is responsible. It is registered under the servlet registry as shown in Listing 3.8. With setting ”/” of the `alias` parameter the download page becomes available under the root path of the Scout server application. For the mapping to the contents `resources/html` the parameter `bundle-path` is

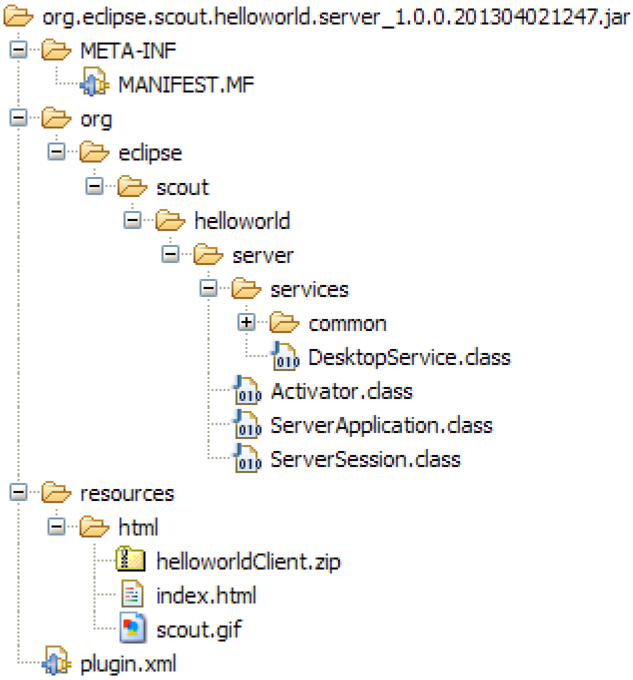
---

[19] See Appendix **??** for more information regarding server-side Equinox.

[20] See Appendix **??** for more information regarding servlets.

[21] See Appendix **??** for more information regarding server-side Eclipse applications (server-side Equinox).

[22] JAR files contain a set of Java classes and associated resources. http://en.wikipedia.org/wiki/JAR_%28file_format%29.



```
org.eclipse.scout.helloworld.server_1.0.0.201304021247.jar
  META-INF
    MANIFEST.MF
  org
    eclipse
      scout
        helloworld
          server
            services
              common
              DesktopService.class
            Activator.class
            ServerApplication.class
            ServerSession.class
  resources
    html
      helloworldClient.zip
      index.html
      scout.gif
  plugin.xml
```

Figure 3.10: The content of the "Hello World" server plugin contained in the `helloworld_server.war` file. The necessary files for the download page including the zipped client application are in the `resources/html` directory.

Listing 3.8: The configuration of the server's resource servlet in the `plugin.xml` configuration file. The remaining content of the file has been omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension
      name=""
      point="org.eclipse.equinox.http.registry.servlets">
    <servlet
        alias="/"
        class="org.eclipse.scout.rt.server.ResourceServlet">
      <init-param
          name="bundle-name"
          value="org.eclipse.scout.helloworld.server">
      </init-param>
      <init-param
          name="bundle-path"
          value="/resources/html">
      </init-param>
    </servlet>
  </extension>

</plugin>
```

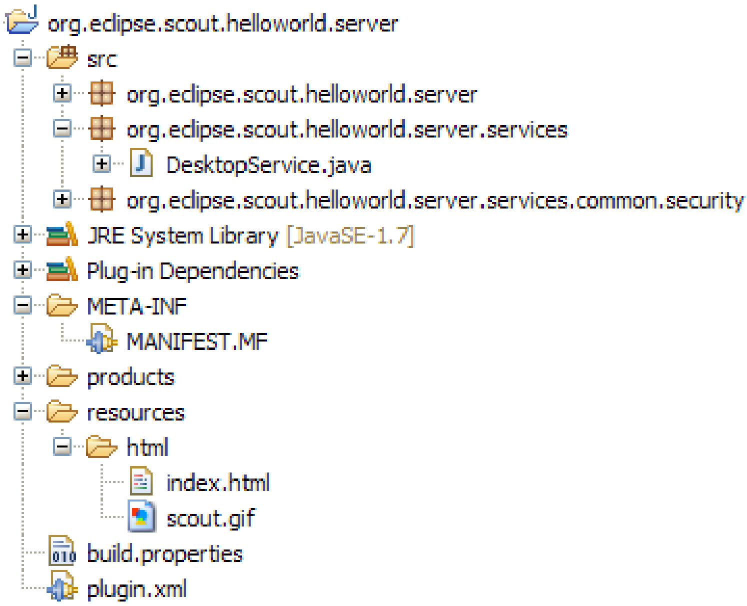

org.eclipse.scout.helloworld.server
src
org.eclipse.scout.helloworld.server
org.eclipse.scout.helloworld.server.services
DesktopService.java
org.eclipse.scout.helloworld.server.services.common.security
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
META-INF
MANIFEST.MF
products
resources
html
index.html
scout.gif
build.properties
plugin.xml

Figure 3.11: The "Hello World" server plugin shown in the Eclipse package explorer. The files for the download page are located under `resources/html`.

used.

Revisiting the "Hello World" server plugin project in the Eclipse package explorer as shown in Figure 3.11, we can see how the plugin project elements are transformed and copied into the JAR file. Examples files are `plugin.xml` and `MANIFEST.MF` as well as static HTML content of the download page (files `index.html` and `scout.gif`). The zipped client is missing of course. It is assembled, zipped and added into the Scout server JAR file by the *Export Scout Project* wizard of the Scout SDK. In case you need to change/brand/amend the download page for the desktop client, you have now learned where to add and change the corresponding HTML files.

3.8 Deploying to Tomcat

In this section we will discuss two common pitfalls when working with the Scout IDE and Tomcat. The symptoms linked to these problems are Scout server applications that are not starting or Scout applications that fail to properly update.

In usual culprit behind Scout server applications that fail to start is a blocked port 8080. This setting can be created when we try to run both the Jetty web server inside the Scout SDK and the local Tomcat instance. In consequence, either Jetty or Tomcat is not able to bind to port 8080 at startup which makes it impossible for a client to connect to the right server. To avoid such conflicts, make sure that you always stop the Scout server application in the Scout SDK (effectively killing Jetty) before you restart your Tomcat server. Alternatively, you can assign two different ports to your Jetty webserver and your Tomcat webserver.

To modify Jetty's port number in the Scout SDK you have to update the corresponding properties in the `config.ini` files of the development products of your Scout server application and all client applications. In the Scout server's `config.ini` file the property is named `org.eclipse.equinox.http.jetty.http.port`, in the client `config.ini` files the relevant property is called `server.url`. To change the port number to 8081 for the ”Hello World” example in the Scout SDK you could use the following lines in the individual `config.ini` files.

Scout Server	org.eclipse.equinox.http.jetty.http.port=8081
Scout Desktop Client	server.url=http://localhost:8081/helloworld_server/process
Scout Ajax Server	server.url=http://localhost:8081/helloworld_server/ajax

The second pitfall is connected to a web application that seems to refuse to update to the content of a freshly generated WAR file. At times it seems that your changes to a deployed WAR file do not find their way to the application actually running. In many cases this is caused by a cached instance of the previous version of your application located in Tomcat's working directory. To save yourself much frustration, it often helps just to clear Tomcat's working directory and restart Tomcat. For this, you may follow the following procedure.

1. Stop the Tomcat web server
2. Go to folder `work/Catalina/localhost`
3. Verify that you are not in Tomcat's `webapps` folder
4. Delete all files and directories in folder `work/Catalina/localhost`
5. Start the Tomcat web server

How you start and stop Tomcat depends on the platform you are running it. If you have installed Tomcat on a Windows box according to Appendix C it will be running as a service. This means that to stop the Tomcat web server you need to stop the corresponding Windows service. For starting and stopping Tomcat on Mac/Linux/Unix systems, you can use the command line script files `startup.sh` and `shutdown.sh` located in Tomcat's subdirectory `bin`.

For those interested in more advanced aspects of Apache Tomcat we recommend the article ”More about the Cat” by Chua Hock-Chuan[23].

---

[23] More about the Cat: `http://www.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_More.html`.

Chapter 4

Scout Tooling

In addition to the Scout runtime framework presented in the previous chapter, Eclipse Scout also includes a comprehensive tooling, the Scout SDK. Thanks to this tooling, developing Scout applications is made simpler, more productive and also more robust. Initially, a solid understanding of the Java language is sufficient to start developing Scout applications and only a rough understanding of the underlying Eclipse/OSGi/JEE technologies is required.

The Scout SDK consists of navigation support for the application model defined by the Scout runtime and provides many intuitive component wizards. This creates an ideal environment to beginners for building complete, high-quality Scout applications. Typically, Java developers only need a few days of Scout training to start creating their own advanced client server business applications.

The Scout SDK also helps developers to become more productive. Many repetitive and error prone tasks run automatically in the background or are taken care of by the component wizards of the Scout SDK. This starts with the initial creation of a Scout client server application, continues with the wizards to create complete dialogs and pages and includes the automatic management of the data transfer objects needed by the client server communication.

Finally, the application code created by the Scout SDK wizards helps to ensure that the resulting Scout application has a consistent and robust code base and is well aligned with the application model defined by the Scout runtime framework.

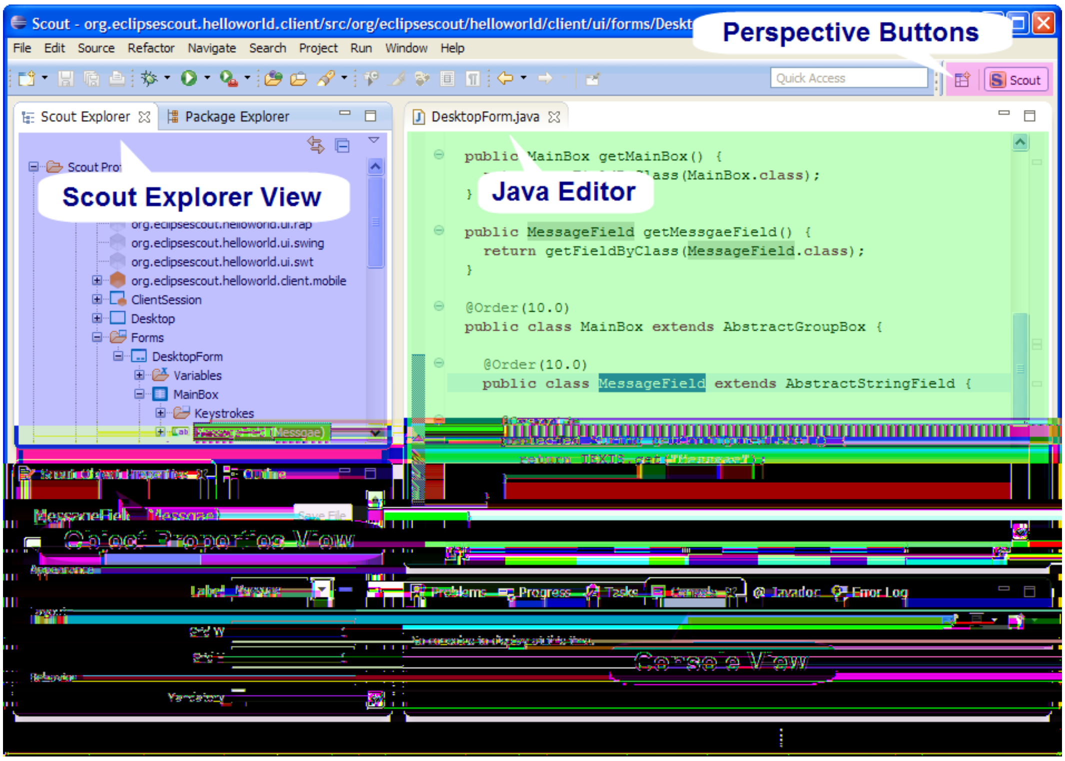
4.1 The Eclipse IDE

To get a good introduction into working with Eclipse as an IDE, we highly recommend the excellent tutorial by L. Vogel[1].

Whatever we will be working with in the Eclipse IDE, we are always working in the context of an Eclipse IDE perspective. An Eclipse IDE perspective contains the visual elements and the arrangement of those elements to support a specific development task within the Eclipse IDE. Perspectives relevant to the development of Scout applications are the Scout perspective, the Java perspective, the Debug perspective, and many others. To open a perspective available in the Eclipse

---

[1] Eclipse IDE Tutorial by Lars Vogel: `http://www.vogella.com/articles/Eclipse/article.html`


Scout - org.eclipsescout.helloworld.client/src/org/eclipsescout/helloworld/client/ui/forms/Deskt
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Scout
Perspective Buttons
Scout Explorer
Package Explorer
Scout Explorer View
org.eclipsescout.helloworld.ui.swing
org.eclipsescout.helloworld.ui.swt
org.eclipsescout.helloworld.client.mobile
ClientSession
Desktop
Forms
DesktopForm
Variables
MainBox
Keystrokes
DesktopForm.java
Java Editor
Outline
Scout Object Properties View
Problems
Progress
Tasks
Javadoc
Error Log
Console View

```
public MainBox getMainBox() {
}

public MessageField getMessgaeField() {
  return getFieldByClass(MessageField.class);
}

@Order(10.0)
public class MainBox extends AbstractGroupBox {

  @Order(10.0)
  public class MessageField extends AbstractStringField {
```

Figure 4.1: The Eclipse IDE with the Scout perspective. The colors indicate the different elements. View parts (blue), editor parts (green) and perspective buttons (purple).

IDE, the Open Perspective button or the Window → Open Perspective → Other... menu can be used.

Figure 4.1 provides a screenshot of the Eclipse Scout perspective indicating the corresponding perspective button and the main view parts and editor parts involved. Using drag and drop, views and editors can be freely moved around in the Eclipse IDE to suit the developer's needs. Perspectives can be further individualized using the Window → Customize Perspective... menu. Here, the visibility of the toolbar items and menu entries can be defined. Once a suitable layout of all desired elements has been defined, this organisation may be saved as a personal perspective using the Eclipse IDE Window → Save Perspective As... menu.

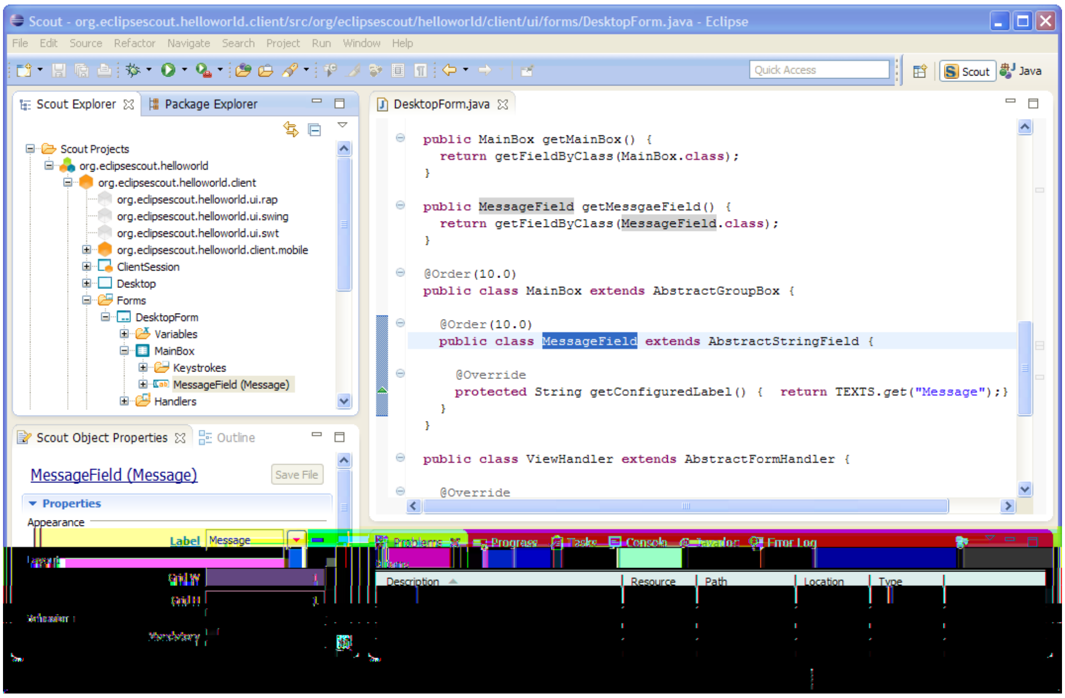
In case a customizing step does not turn out as intended, with the Window → Reset Perspective... menu is always possible to go back to the last saved state of the current perspective.

4.2 The Scout SDK

The Scout SDK is added to the Eclipse IDE in the form of the Scout perspective[2]. With the Scout Explorer and Scout Objects Properties, two view parts are contained in the Scout perspective. Additionally, the Scout SDK contains a comprehensive set of wizards that support the developer in creating Scout application components.

---

[2] See Appendix ?? for additional information regarding Eclipse IDE perspectives.


Scout - org.eclipsescout.helloworld.client/src/org/eclipsescout/helloworld/client/ui/forms/DesktopForm.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Scout
Java
Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.helloworld
org.eclipsescout.helloworld.client
org.eclipsescout.helloworld.ui.rap
org.eclipsescout.helloworld.ui.swing
org.eclipsescout.helloworld.ui.swt
org.eclipsescout.helloworld.client.mobile
ClientSession
Desktop
Forms
DesktopForm
Variables
MainBox
Keystrokes
MessageField (Message)
Handlers
Scout Object Properties
Outline
MessageField (Message)
Save File
Properties
Appearance
Label
Message
DesktopForm.java
Problems
Progress
Tasks
Console
Javadoc
Error Log
Description
Resource
Path
Location
Type

```
public MainBox getMainBox() {
  return getFieldByClass(MainBox.class);
}

public MessageField getMessgaeField() {
  return getFieldByClass(MessageField.class);
}

@Order(10.0)
public class MainBox extends AbstractGroupBox {

  @Order(10.0)
  public class MessageField extends AbstractStringField {

    @Override
    protected String getConfiguredLabel() {  return TEXTS.get("Message");}
  }
}

public class ViewHandler extends AbstractFormHandler {

  @Override
```

Figure 4.2: The Scout SDK perspective. On the left hand side the Eclipse Scout Explorer and the Scout Object Properties view are visible.

The Scout Explorer view allows the developer to navigate the complete Scout application model. Once an element in the Scout Explorer is selected, the Scout Object properties view allows to view and change selected properties of the selected element. Depending on the selected element in the Scout Explorer, the Scout SDK offers appropriate context menus to start the related wizards.

Figure 4.2 provides a screenshot of the Scout SDK perspective. In the Eclipse Scout Explorer shown in the upper left part of the screenshot, the message field in the desktop form of the "Hello World" application is selected. In the Scout Object Properties located below, the message field's appearance, layout and behaviour properties are displayed. On the right side, the corresponding source code is loaded in a Java editor.

When the developer changes a property of the selected element, the Java code is updated accordingly. For example, clicking the *Mandatory* property in the Scout Object Properties of the message field will insert the method `getConfiguredLabel` to the message field's class. This demonstrates how the Scout SDK directly works on the Java source code. In fact, the Java source code is the only artefact relevant for the Scout SDK to 'understand' the Scout application model. Taking advantage of this setup, the Scout SDK implements a full round-trip-engineering from creating the Java code for Scout application components, parsing code changes in the background, and displaying the current implementation of the Scout application in the Scout Explorer and the Scout Object Properties.

Thanks to the round-trip-engineering provided by the Scout SDK, the information presented in the Scout Explorer and the Scout Object Properties always stay in sync with the Java code of the Scout application. To illustrate this, we will re-use the "Hello World" Scout project from Chapter 2. Start the Eclipse IDE with the workspace containing the "Hello World" application.

Then, navigate to method `getConfiguredLabel` as shown in Figure 4.2, and add the java snippet shown below to the class `MessageField`.

```
@Override
protected boolean getConfiguredMandatory() {
  return true;
}
```

After having saved the code change, you can observe that the *Mandatory* property in the section Behaviour of the message field's Scout Object properties has changed its state. The font of its label is now presented in bold face and underlined, the checkbox is ticked and a red minus icon is shown on the right side of the property. Obviously, the Scout SDK is directly operating on the project's source code and does not rely or need any external meta data. This provides the flexibility to develop Scout applications with or without the support of the Scout SDK. And this choice offered to the Scout developer is one of the most important features provided by the Scout SDK. The Scout developer may take advantage of the development support provided by the Scout SDK without being restricted by the Scout tooling in any way.

Technically, the Scout SDK is a set of Eclipse plugins that operate on top of the Eclipse JDT and the Eclipse PDE projects. The Java Development Tools (JDT)[3] contain a Java IDE supporting the development of any Java application, and the Plugin Development Environment (PDE)[4] provides tools to create, develop, test, debug, build and deploy Eclipse plugins, and additional artefacts relevant for Eclipse based applications. As in the case of the Scout Runtime, the plugins representing the Scout SDK, the JDT and the PDE are all located in the `plugins` directory of your Eclipse installation and named `org.eclipse.scout.sdk.*`, `org.eclipse.jdt.*` and `org.eclipse.pde.*` .

4.3 The Scout Explorer

The Scout Explorer view is responsible for the navigation support within the Scout application model. This navigation support is presented in the form of a tree view and includes the client with its UI components, the server and the shared part of a Scout application. It also includes all Scout application modules of modular Scout applications[5]. For the actual navigation in the tree representing the Scout application both the mouse or the keyboard can be used.

To expand or collapse a selected node in the Scout Explorer, you may click on the tiny PLUS icon or the MINUS icon presented to the left of the node. Alternatively, you can also use the <RIGHT> or the <LEFT>.

Once a node in the tree is selected, the Scout Object Properties view presents the associated configuration of the selected element. If the selected element represents a specific application model component, the corresponding Java source code can be accessed through a double click on the node, or hitting the <ENTER>.

The navigation tree provided in the Scout Explorer view also allows the developer to add elements to your application. Depending on the selected node in the tree, the Scout SDK allows to start the applicable Scout SDK wizards using corresponding context menus. The wizards support

---

[3] See the Eclipse JDT project page for details: `http://www.eclipse.org/jdt/`.

[4] See the Eclipse PDE project page for details: `http://www.eclipse.org/pde/`.

[5] See Section **??** for more information regarding multi module Scout applications.

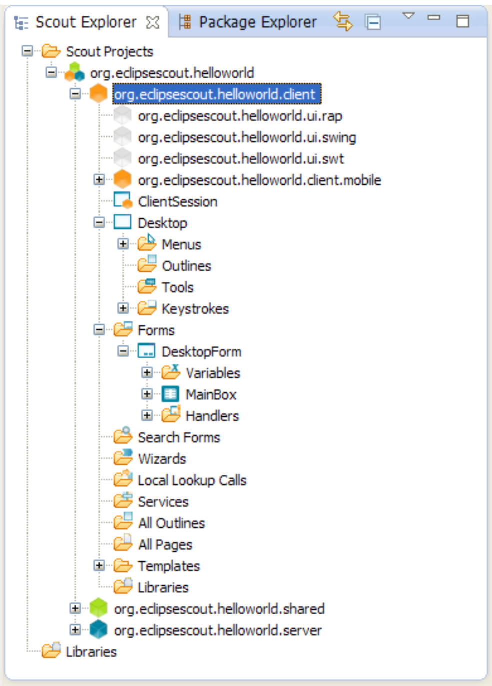

Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.helloworld
org.eclipsescout.helloworld.client
org.eclipsescout.helloworld.ui.rap
org.eclipsescout.helloworld.ui.swing
org.eclipsescout.helloworld.ui.swt
org.eclipsescout.helloworld.client.mobile
ClientSession
Desktop
Menus
Outlines
Tools
Keystrokes
Forms
DesktopForm
Variables
MainBox
Handlers
Search Forms
Wizards
Local Lookup Calls
Services
All Outlines
All Pages
Templates
Libraries
org.eclipsescout.helloworld.shared
org.eclipsescout.helloworld.server
Libraries

Figure 4.3: The Scout Explorer view. The white nodes below the expanded client node represent the supported UI technologies.

the creation of application components, such as dialogues on the client side or services on the server side by generating the necessary Java code.

In Figure 4.3 the top level organisation of the client application model is shown as it is represented in the Scout Explorer. All client specific elements are located under the selected orange client node *org.eclipsescout.helloworld.client*. Right below, the three white UI plugins are located that represent the support for the corresponding UI technologies for Swing, SWT and Eclipse RAP. The orange *org.eclipsescout.helloworld.client.mobile* node contains all elements that are specific to mobile devices such as the `MobileHomeForm`

Specific nodes for the client session and the desktop of the Scout client allow access to the corresponding Scout application model components. While the client session is the main entry point for client-server communication, the desktop represents the root component of the visible part of a Scout client applications. Below, a set of folders group additional client model components according to their type. The forms folder for example holds all available forms, such as the desktop form that we have seen in the "Hello World!" tutorial[6].

A screenshot of the expanded green shared node *org.eclipsescout.helloworld.* is provided in Figure 4.4. As the name "shared" suggests, the corresponding plugin holds all application components that are required for both the Scout client and the Scout server application. This includes texts, icons, enumeration or codes, permissions, lookup calls. As shown in Figure 4.4, a separate folder for each resource type is provided under the shared node.

In Figure 4.5, the blue server node is expanded in the Scout Explorer view. As the primary responsibility of the Scout server application is dealing with requests, its components are mostly

---

[6] See Section 3.2 for a description of many of these elements.

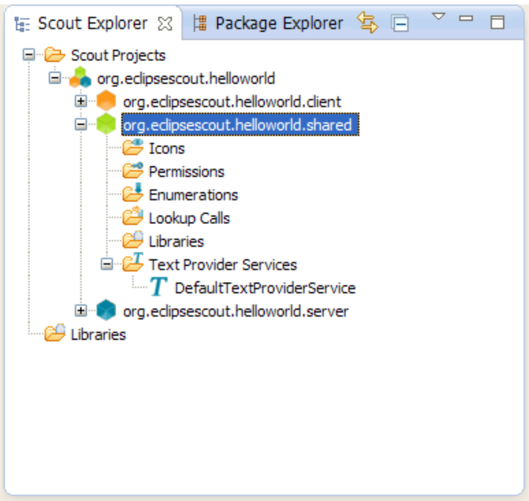

Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.helloworld
org.eclipsescout.helloworld.client
org.eclipsescout.helloworld.shared
Icons
Permissions
Enumerations
Lookup Calls
Libraries
Text Provider Services
DefaultTextProviderService
org.eclipsescout.helloworld.server
Libraries

Figure 4.4: The Scout Explorer view with the expanded shared node.

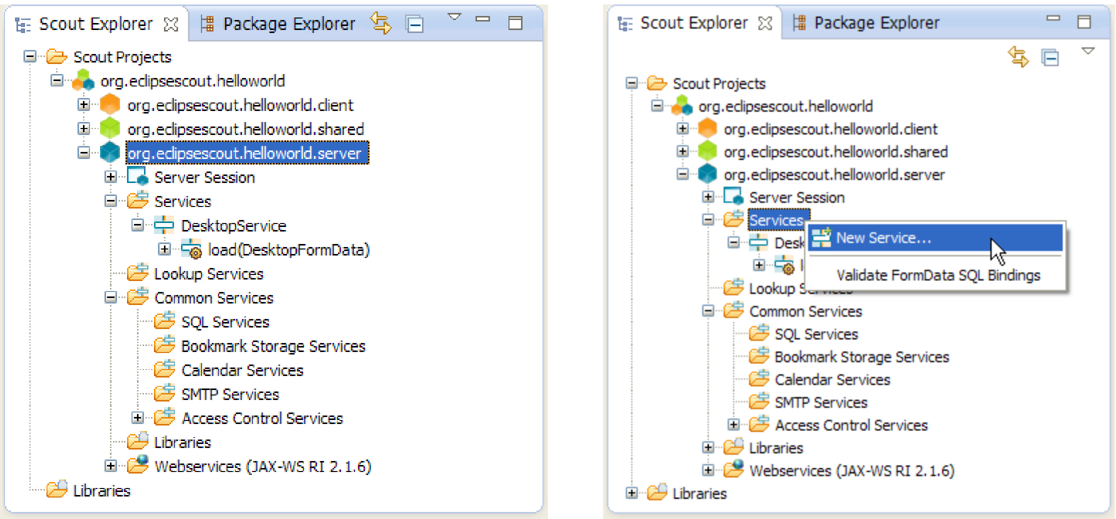


Figure 4.5: The Scout Explorer view with the expanded server node (left). On the individual nodes, access to the relevant Scout SDK wizards is provided via context menus (right).

related to different types of services. Below the server session node, the *Services* folder holds services related to the processing logic of the application such as retrieving and updating data. The remaining folder group different more specific types of server services. Under the *Webservices* folder the Scout SDK support to provide and consume web services is located.

The right side of Figure 4.5 illustrates the access to the Scout SDK wizards via corresponding context menus. The NEW SERVICE... menu shown in the screenshot will start the wizard to add a new Scout server service.

The different colored tree nodes discussed above are all represented by their individual Eclipse plugins. This includes the orange client node, the white UI nodes, the orange mobile client node, the green shared node and the blue server node. A Scout Swing client for example contains the plugins *org.eclipsescout.helloworld.client*, *org.eclipsescout.helloworld.shared* and *org.eclipsescout.helloworld.client.ui.swing* but not the other UI technology plugins. The Scout server contains the *org.eclipsescout.helloworld.server* plugin and the *org.eclipsescout.helloworld.shared* plugin.

4.4 The Scout Object Properties

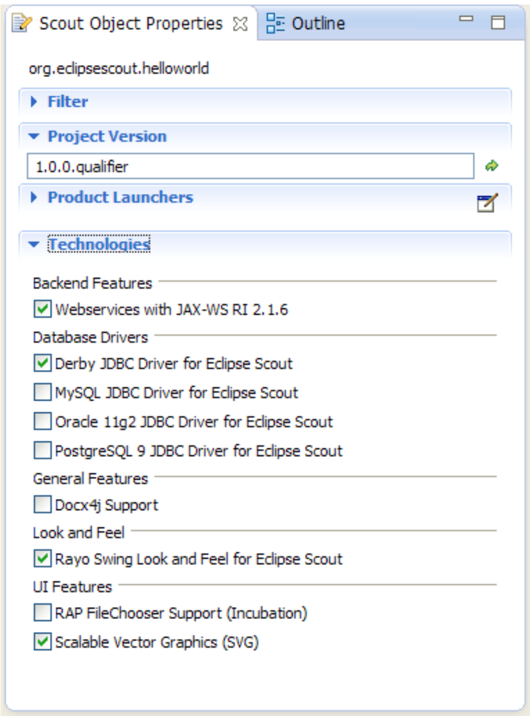
The Scout Object Properties view provides direct access to configurable properties and operations for the element selected in the Scout Explorer. Before we discuss the typical layout of an object property view we describe the special case of the property view for a complete Scout application. This property view is displayed when the application's top level node is selected in the Scout Explorer as shown in Figure 2.3.

The main elements of the top-level application properties are the sections *Product Launchers* and the *Technologies*. As the product launcher section with its launcher boxes has already been covered in Section 2.3 and Section 3.3 we focus on the technologies section here. The technologies section allows to add Scout runtime features and functionalities to the Scout application or remove such elements from the application.

When the selection of a technology checkbox is changed, a message box is shown to the user. This box lists all project resources that are changed when the user confirms the action. Once the dialog is confirmed, the selected resources are modified by the Scout SDK to add or remove the feature.

For features containing licences not compatible to the Eclipse Public Licence (EPL) or features in incubation status, the necessary artefacts are not provided as part of the Eclipse Scout installation package. Instead, the associated artefacts need to be installed from a remote updated site first. Before any non-EPL content is downloaded from the internet, the user needs to review and confirm the associated licence. For this, a license confirmation dialog is shown upfront. After confirmation, the required files are downloaded and automatically installed in the local Eclipse instance of the developer. Usually, the Eclipse IDE needs to restart after a feature is downloaded and installed for the first time. Currently, the procedure described above is used for the following technologies.

- MySQL JDBC Driver for Eclipse Scout
- Oracle 11g2 JDBC Driver for Eclipse Scout
- PostgresSQL 9 JDBC Driver for Eclipse Scout



org.eclipsescout.helloworld

Filter

Project Version

1.0.0.qualifier

Product Launchers

Technologies

Backend Features

☑ Webservices with JAX-WS RI 2.1.6

Database Drivers

☑ Derby JDBC Driver for Eclipse Scout
☐ MySQL JDBC Driver for Eclipse Scout
☐ Oracle 11g2 JDBC Driver for Eclipse Scout
☐ PostgreSQL 9 JDBC Driver for Eclipse Scout

General Features

☐ Docx4j Support

Look and Feel

☑ Rayo Swing Look and Feel for Eclipse Scout

UI Features

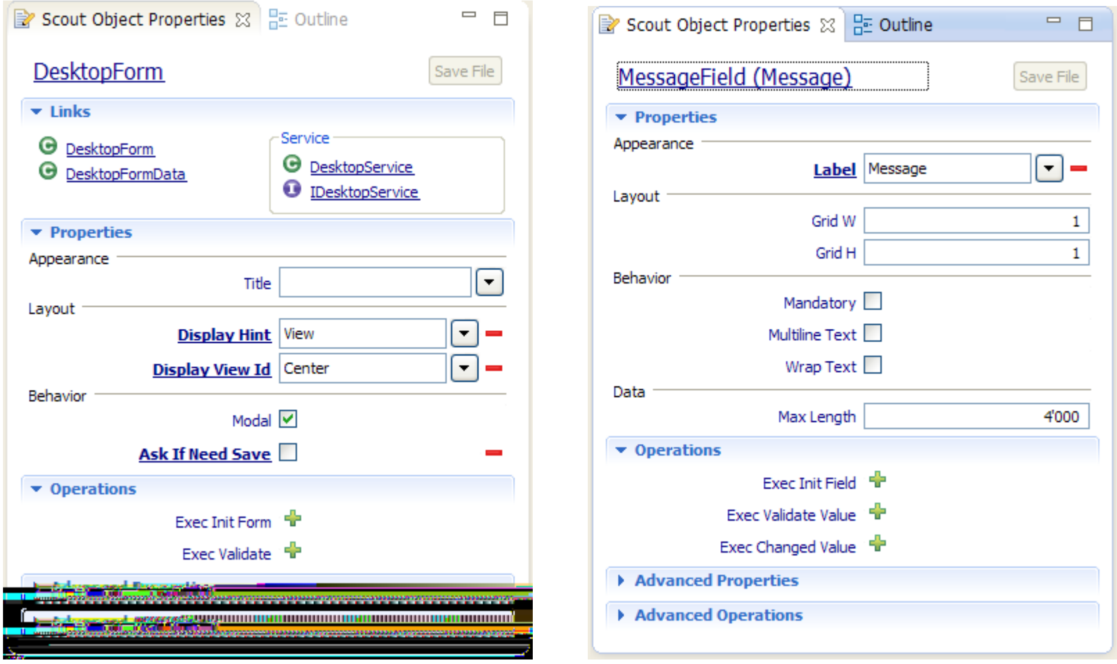
☐ RAP FileChooser Support (Incubation)
☑ Scalable Vector Graphics (SVG)

Figure 4.6: The Scout Object Properties showing the expanded Technologies section.

- Docx4j Support
- Rayo Swing Look and Feel for Eclipse Scout
- RAP FileChooser Support (Incubation)

For the description of the Scout Object Properties of typical Scout components we use the example views provided in Figure 4.7. Both Scout Object Properties example views for the desktop form and the message field are take from the "Hello World!" application described in Chapter 2. As in the case of the top-level node representing the complete Scout application, the typical layout of the Scout Object Properties view is organized into several sections. The content and ordering of the property sections always follows the same scheme.

- Filter
- Links
- Properties
- Operations
- Advanced Properties
- Advanced Operations



DesktopForm

DesktopForm
DesktopFormData
DesktopService
IDesktopService

Appearance
Title

Layout
Display Hint View
Display View Id Center

Behavior
Modal ☑
Ask If Need Save ☐

Exec Init Form
Exec Validate

MessageField (Message)

Appearance
Label Message

Layout
Grid W 1
Grid H 1

Behavior
Mandatory ☐
Multiline Text ☐
Wrap Text ☐

Data
Max Length 4'000

Exec Init Field
Exec Validate Value
Exec Changed Value

Figure 4.7: The Scout Object Properties for a complete form (left) and a string form field (right).

If a folder node (such as the *Forms* folder under the orange client node), is selected in the Scout Explorer, the Scout Object Properties only shows a filter section with a filter field. The content in this filter field then restricts the elements below the folder icon accordingly. This feature is especially useful for the development of larger applications containing dozens of forms or services.

The *Links* section provides a set of hyperlinks as shown on the left hand side of Figure 4.7. The provided links all refer to Java classes and interfaces that related to the Scout component represented by the property view.

The available properties of a Scout component are listed in sections *Properties* and *Advanced Properties*. Basically, all available Java `getConfigured` methods of a Scout component are listed in one of the two sections depending on the frequency of their usage. Section *Properties* shows the often used properties, while the less frequently used properties are provided in section *Advanced Properties*. The examples in Figure 4.7 show the thematic organization of the sections into *Appearance*, *Layout*, *Behaviour* and other groups.

Access to object operations implemented with Java `exec` methods is provided in sections *Operations* and *Advanced Operations*. Again, the more and less frequently used operations are assigned to individual sections.

For all listed properties and operations the corresponding Javadoc is displayed in the form of a tooltip window. To display the available Javadoc, move the mouse pointer over the method of interest in the Scout Object Properties[7].

So far, we did only describe the content and organisation of the Scout Object Properties. In the reminder of this section we will describe the procedure to change or update the default values of

---

[7] This features is work in progress, as for some methods shown in the Scout Object Properties the Javadoc content is not yet available.

properties or the default behaviour of Scout components. To indicate non-default property values or non-default behaviour, the font of the property or the operation is switched to underlined bold face. As an example, consider the property *Ask If Need Save* of the desktop form shown on the left side of Figure 4.7. The bold font visualizes that this does not correspond to the default behaviour of forms. And underlining of this property further indicates, that the label has become a clickable link. Clicking on this label will load the corresponding method into the Java editor in the Scout SDK.

To change a property for a specific Scout component just enter a value in the corresponding field or tick/untick the provided checkbox. In some cases the value may also be chosen from a dropdown list. For the label field shown on the right hand side of Figure 4.7, the "Message" value refers to the text key to be used in method `getConfiguredLabel`. To enter a new label text, start typing the desired translated label and pick the option "New translated text...".

To reset a property or operation to its default value/behaviour, you may click on the red MINUS icon provided on the right side of the property field. This will remove the overridden method from the object's Java code.

4.5 Scout SDK Wizards

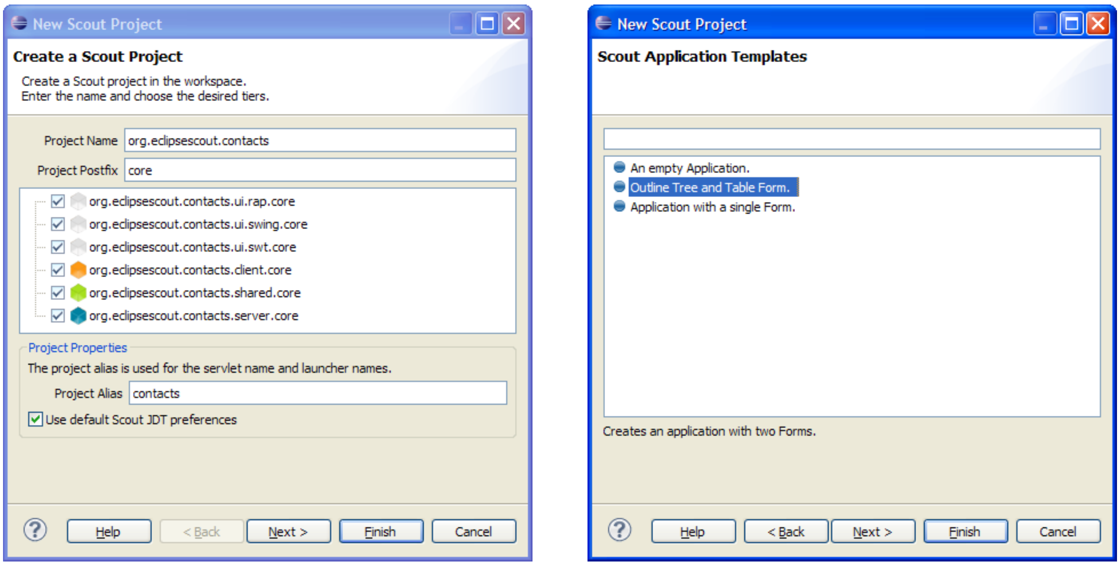
The Scout SDK wizards allow the developer to create Scout application model components with just a few clicks. Creation wizards are provided for all major model components such as forms on the client side and services on the server side. The usage of wizards not only makes developing Scout applications efficient but also helps to create robust code and reduces the number of errors.

Scout SDK wizards can do many things. They add and update Java classes, register services in the `plugin.xml` files, manage plugin dependencies in the `MANIFEST.MF` files and update Eclipse product files when necessary. All these capabilities hide a lot of the complexity of writing Eclipse based applications. This simplifies the developing of Scout application considerably.

In the text below, descriptions of the most commonly used Scout SDK wizards are provided. First, the wizards to create and export complete Scout applications are described. Then, the wizards to create Scout application model components are introduced.

4.5.1 Creating a new Scout Project

The *New Scout Project* wizard allows to create a new Scout client server application. In the Scout Explorer, the NEW SCOUT PROJECT... context menu is provided on the top-level *Scout Projects* folder. The creation of a form based Scout application has already been introduced in Section 2.2 for the "Hello World" tutorial. And Sectionprd 0.9pthisthNjooud" ienScou88(re3("New)-Hell(for)ther)-355ed. Then



New Scout Project

Create a Scout Project

Create a Scout project in the workspace.
Enter the name and choose the desired tiers.

Project Name: org.eclipsescout.contacts
Project Postfix: core

- ☑ org.eclipsescout.contacts.ui.rap.core
- ☑ org.eclipsescout.contacts.ui.swing.core
- ☑ org.eclipsescout.contacts.ui.swt.core
- ☑ org.eclipsescout.contacts.client.core
- ☑ org.eclipsescout.contacts.shared.core
- ☑ org.eclipsescout.contacts.server.core

Project Properties

The project alias is used for the servlet name and launcher names.

Project Alias: contacts

☑ Use default Scout JDT preferences

New Scout Project

Scout Application Templates

- ○ An empty Application.
- ● Outline Tree and Table Form.
- ○ Application with a single Form.

Creates an application with two Forms.

Figure 4.8: The Scout SDK wizard to create a new Scout application. The first wizard dialog (left) is used to specify the needed application plugins. In the next wizard step (right) the application template is selected.

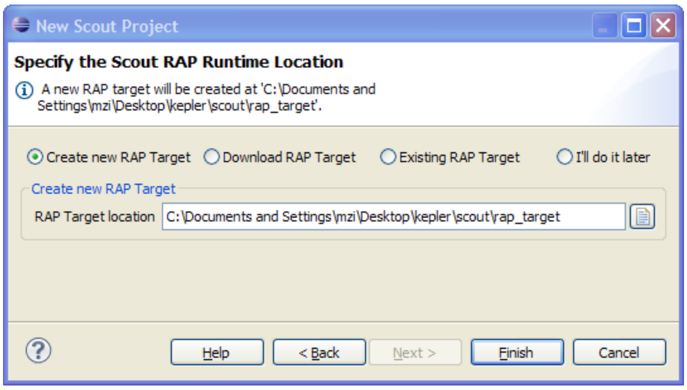
cation plugins (UI, client, shared, server) are then named following the scheme `<Project Name>.<plugin>.<Project Postfix>`.

In the box below the postfix field the list of application plugins that may be created by this wizard is provided. Initially, all possible plugins are checked which is useful to quickly demonstrate a ”Hello World” application. However, for a typical setup only one or two UI plugins will be needed. To develop a mobile/web application only the RAP UI plugin is necessary. For a desktop only client, either the Swing or the SWT UI plugin will be needed. And in cases where the client needs to run both on the desktop and as a web/mobile application, the RAP and a desktop UI plugin can be chosen. A Scout application must not necessarily be a client-server application. It is also possible to create a client only or a server only application. But make sure to include the shared plugin in all possible use cases.

The content of the *Alias* field is used for the client’s executable file, the name of the servlet representing the Scout server application and to build the product launcher names. And if the *Use default Scout JDT preferences* field is checked, the Scout default Java development settings are used. Otherwise, you start with no settings and can apply your own template.

As we have seen in the ”Hello World” tutorial, it is sufficient to provide a project name in the first wizard step and click on the FINISH button to create a form based client server application. But we can also click on the NEXT button to choose an application template.

The second wizard step shown on the right side of Figure 4.8 allows to choose a Scout application template for the Scout project. When choosing the template *Empty Application*, you exactly know what you do and only a minimal code base is created. The other two application templates represent different application types. The template *Application with a single Form* is the default choice used for the ”Hello World” tutorial. Finally, the template *Outline Tree and Table Form* can be used to build explorer like applications. The outline tree is typically used to navigate between related business entities. And the tables are used to list a number of business entities with their



Specify the Scout RAP Runtime Location

A new RAP target will be created at 'C:\Documents and Settings\mzi\Desktop\kepler\scout\rap_target'.

● Create new RAP Target ○ Download RAP Target ○ Existing RAP Target ○ I'll do it later

Create new RAP Target

RAP Target location: C:\Documents and Settings\mzi\Desktop\kepler\scout\rap_target

Figure 4.9: The last wizard step is used to specify the RAP target for the new Scout application.

attributes. We will use this template for the creation of a larger Scout application in Chapter 5. As in the case of the first wizard step, you may click FINISH button to start the creation of an initial Scout application. Or, click on NEXT button and manually step through the third and last wizard step.

The third wizard step shown in Figure 4.9 is only available if the RAP UI plugin has been checked in Step 1. Because the RAP runtime must not be installed into the running Eclipse instance[8], a separate target platform is created and/or used by the Scout SDK. This target platform then contains all necessary plugins to run the Scout RAP UI. The different options provided by this wizard step are the following.

When choosing the option *Create new RAP Target*, a new RAP target platform will be created at the location specified. This target platform is then created by the Scout SDK in the given directory and used by all application plugins.

Option *Download RAP Target* will downloaded the target platform into the running workspace. This download will then only be available to the active workspace. The two download types are only download the RAP plugins (checkbox not ticked) and download a new Kepler Eclipse platform as well (checkbox ticked).

With the option *Existing RAP Target* an existing RAP target location can be specified. The wizard then detects whether the provided location contains a complete target platform or only the RAP target plugins. If a complete platform is detected, only the directory specified is used as the target platform. Otherwise, the plugins available to the running Eclipse and the content of the provided directory jointly define the target platform.

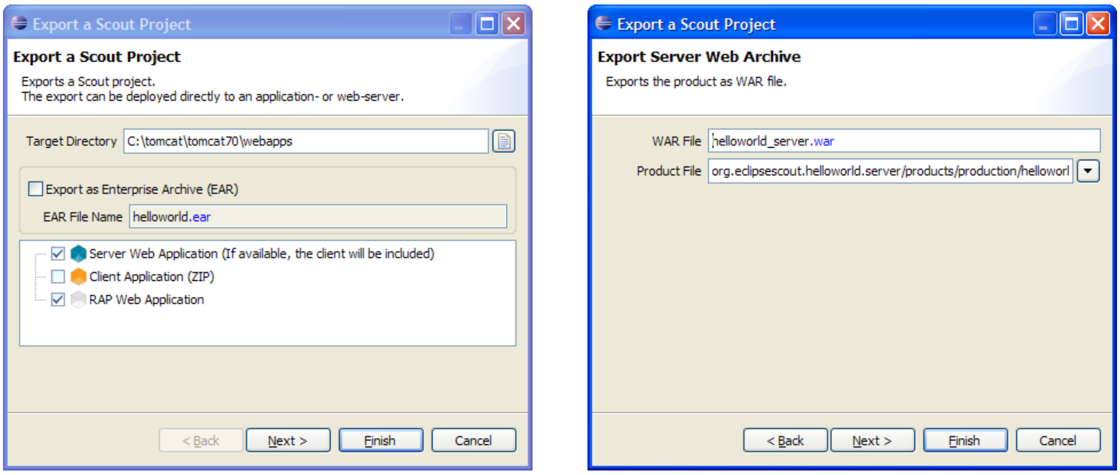
To manually define a RAP target platform choose option *I'll do it later*. With this option, the Scout SDK does not create a RAP target platform for you. But note that the created project will not compile before a complete target platform has been created for the Scout application.

4.5.2 Exporting a Scout Project

The *Export a Scout Project* wizard allows to export a complete Scout client server application as WAR files. In the Scout Explorer, the EXPORT SCOUT PROJECT... context menu is provided on the main application node just below the top-level *Scout Projects* folder.

---

[8] Why?



Export a Scout Project

Exports a Scout project.
The export can be deployed directly to an application- or web-server.

Target Directory: C:\tomcat\tomcat70\webapps

☐ Export as Enterprise Archive (EAR)

EAR File Name: helloworld.ear

☑ Server Web Application (If available, the client will be included)
☐ Client Application (ZIP)
☑ RAP Web Application

Export Server Web Archive

Exports the product as WAR file.

WAR File: helloworld_server.war

Product File: org.eclipsescout.helloworld.server/products/production/helloworl

Figure 4.10: The Scout SDK wizard to a new Scout application into WAR files. The first wizard step (left) is used to select the artefact to be exported. In the next wizard step (right) is used to define the server WAR file name and to select the server product file to be used for the export.

As a simple use case, the usage of the export wizard is described in Section 2.7 of the "Hello World" tutorial. And the corresponding background Section 3.7 provides information regarding the content and the organisation of the WAR files produced by this wizard.

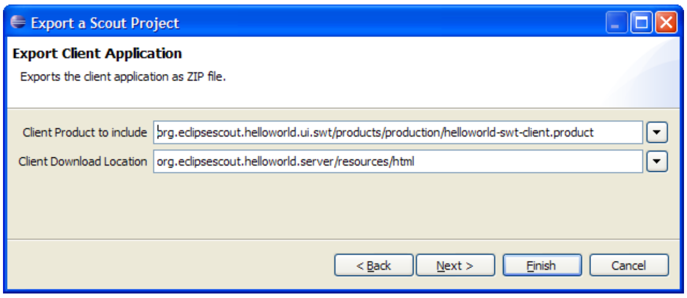
In the first export wizard step shown on the left hand side of Figure 4.10 the target directory and the type of content to be exported is defined. The *Target Directory* field is used to define the directory where the generated WAR files will be exported to. Checking the *Export as Enterprise Archive (EAR)* field the WAR file(s) will be packed into a EAR file using the file name provided in the *EAR File Name* field.

In the artefact selector box the content to be included in the export can be specified. The blue *Server Web Application* node represents the Scout server application. If ticked, the corresponding WAR file will also include a zipped desktop client if such a client is defined in the current Scout project. Selecting the orange *Client Application (ZIP)* node node will put a zipped client into the target directory. Finally, when working with web/mobile applications the necessary RAP server is represented by the white *RAP Web Application* node.

The default scenario assumes that you work with a Scout client server application including a SWT desktop client and corresponding web/mobile clients. For this setup you just need to provide a target directory before you can click the FINISH button to start the export. To verify/update what the export wizard will create you may click the NEXT button to move to the second wizard step.

The second wizard step is shown on the right side of Figure 4.10. The server WAR file name proposed in the *WAR File* field uses the naming scheme `<ProjectAlias>_server.war`. In order for this WAR file to work out-of-the-box it it is recommended to use the proposed value. In the *Product File* field the server product file to be used for the creation of the WAR file can be specified. Clicking the NEXT button will move to the next wizard step to select the client product to be exported.

In the third wizard step shown in Figure 4.11 the client product file can be selected. If the current Scout project defines both a Swing and a SWT client, the default will use the SWT



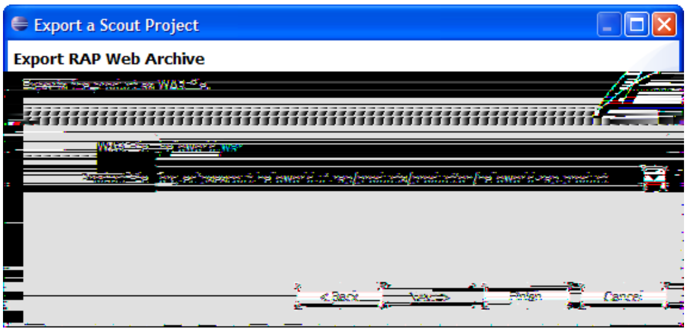
Export Client Application

Exports the client application as ZIP file.

Client Product to include: org.eclipsescout.helloworld.ui.swt/products/production/helloworld-swt-client.product

Client Download Location: org.eclipsescout.helloworld.server/resources/html

Figure 4.11: The third export wizard step is used to specify the desktop client product file to be used for the export and the download location for the resulting zipped client.



Export RAP Web Archive

Figure 4.12: The last export wizard step is used to specify the define the name of the server WAR for the RAP web/mobile application and the corresponding product file to be used for the export.

client. Clicking on the dropdown button next to the *Client Product to include* field will open a *Select Product* dialog to choose the right product from all available client products. In the *Client Download Location* field the path to the zipped client inside the server WAR file is defined. In order for this WAR file to work out-of-the-box it is recommended to use the proposed value. With the NEXT button the last export wizard step is shown.

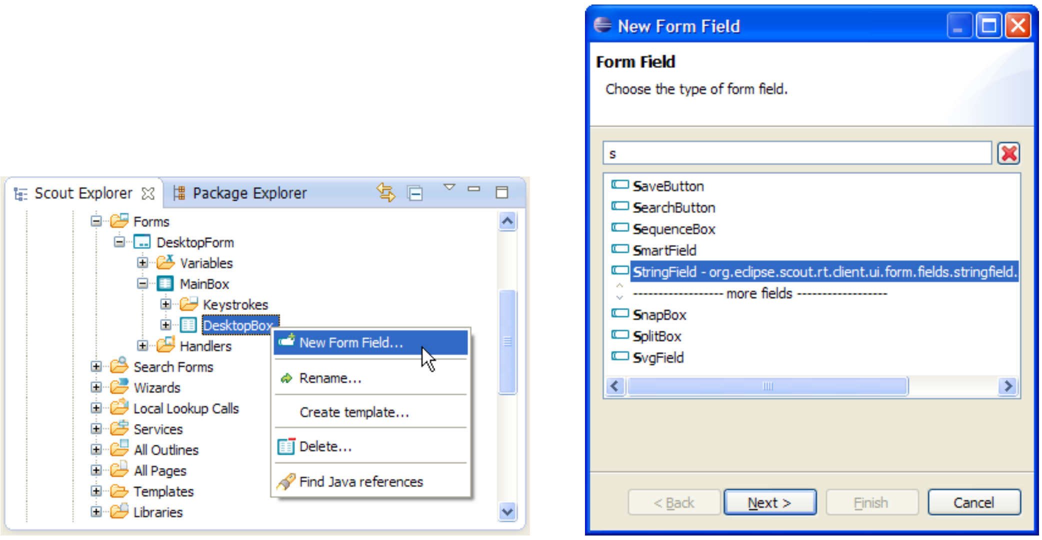
The last export wizard step is shown in Figure 4.12. A RAP server WAR file name is proposed in the *WAR File* field based on the naming scheme `<ProjectAlias>.war`. In order for this WAR file to work out-of-the-box it is recommended to use the proposed value. In the *Product File* field the RAP server product file to be used for the creation of the WAR file can be specified. The FINISH button will then start the export wizard with the settings verified and/or updated in all export wizard steps.

After the wizard has completed the export, all resulting artefacts will be located in the target directory specified in the first wizard step.

4.5.3 Creating new Form Fields

The *New Form Field* wizard allows to create a new form field. In the Scout Explorer, the NEW FORM FIELD... context menu is provided on Scout Explorer nodes representing composite form fields, such as the *MainBox* node.

Figure 4.13 provides screenshots for starting the form field wizard with the context menu and



Choose the type of form field.

s

Figure 4.13: Starting the new form field wizard with the context menu on a composite field (left). The first wizard step (right) is used to select the field type.

the first wizard page. In the first wizard step, the list of all available Scout form field types is presented. To quickly find the desired form field type, a part of the type name can be entered in the search field above the field type list. In the screenshot on the right hand side of Figure 4.13 this search field contains the value 's' which filters the field type list accordingly. Once the type of the new form field is selected, the next wizard step can be loaded by clicking on the NEXT button.

In the second wizard step shown in Figure 4.14 the remaining creation parameters for the new form field can be specified. The *Name* field is used to define the content of the field label. For this, a translated text entry has to be defined. By typing a string into the name field, all potentially matching existing translation entries are shown in a dropdown list. As shown in Figure 4.14 not only the text key is used to find matching entries but also translated text entries. When assigning the label for a preferences string field, the substring "pref" lists a set of text keys that have a matching translated text entry.

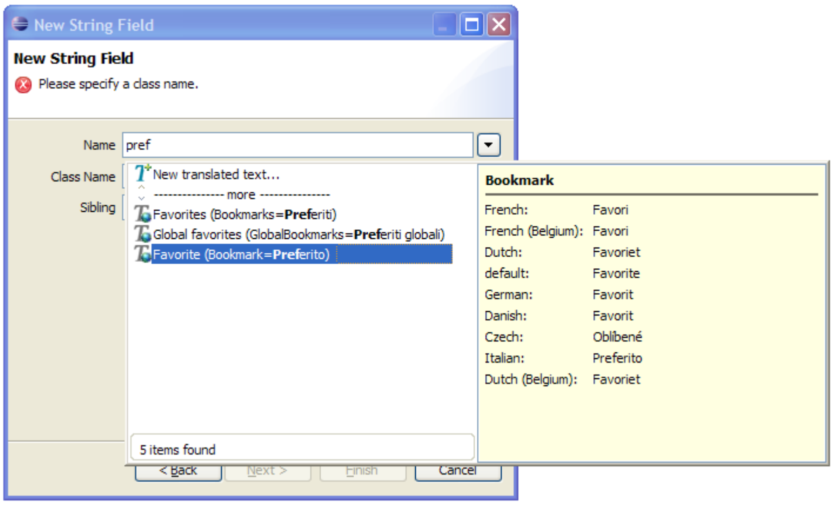
One of the presented options in the list of text entries is the *New translated text...* entry. A double click on this entry starts the wizard to create a new text entry as shown on the left hand side of Figure 4.15. The *Key Name* field holds the text key that is used to access translated text[9]. In the tabs below the key name field, the text translations for the registered languages can be provided. Make sure to at least provide a text in the *default* tab. This text will be used in the Scout application if no translation is available that better matches the logged in user's locale.

Once the name field is filled in, the entered text key is also used to create a proposal for the *Class Name* field using the pattern `<Field Name>Field`. In contrast to the label field, the class name field is mandatory. If the field does not need a label the name field can remain empty. In that case, a class name still needs to be provided as described in Section 2.4 for the `DesktopBox` class in the "Hello World" application.

The *Sibling* field is used to define the ordering of the form fields in the parent container. As

---

[9] The access to the translated texts using `TEXTS.get` is described in the "Hello World" UI background chapter in Section 3.4.



New String Field

Please specify a class name.

Name: pref

Class Name

Sibling

New translated text...

--------------- more ---------------

Favorites (Bookmarks=**Preferiti**)

Global favorites (GlobalBookmarks=**Preferiti** globali)

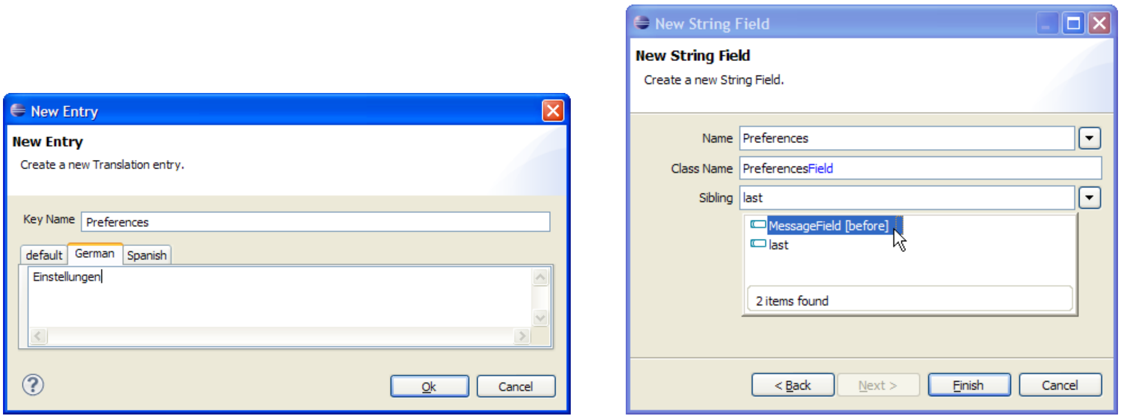
Favorite (Bookmark=**Preferito**)

5 items found

Bookmark

French:	Favori
French (Belgium):	Favori
Dutch:	Favoriet
default:	Favorite
German:	Favorit
Danish:	Favorit
Czech:	Oblibené
Italian:	Preferito
Dutch (Belgium):	Favoriet

Figure 4.14: The second form field wizard step is also used to assign a field label. By typing a word or a part of a word into the *Name* field a list of existing and matching text entries is provided.



New Entry

Create a new Translation entry.

Key Name: Preferences

Einstellungen

New String Field

Create a new String Field.

Name: Preferences

Class Name: PreferencesField

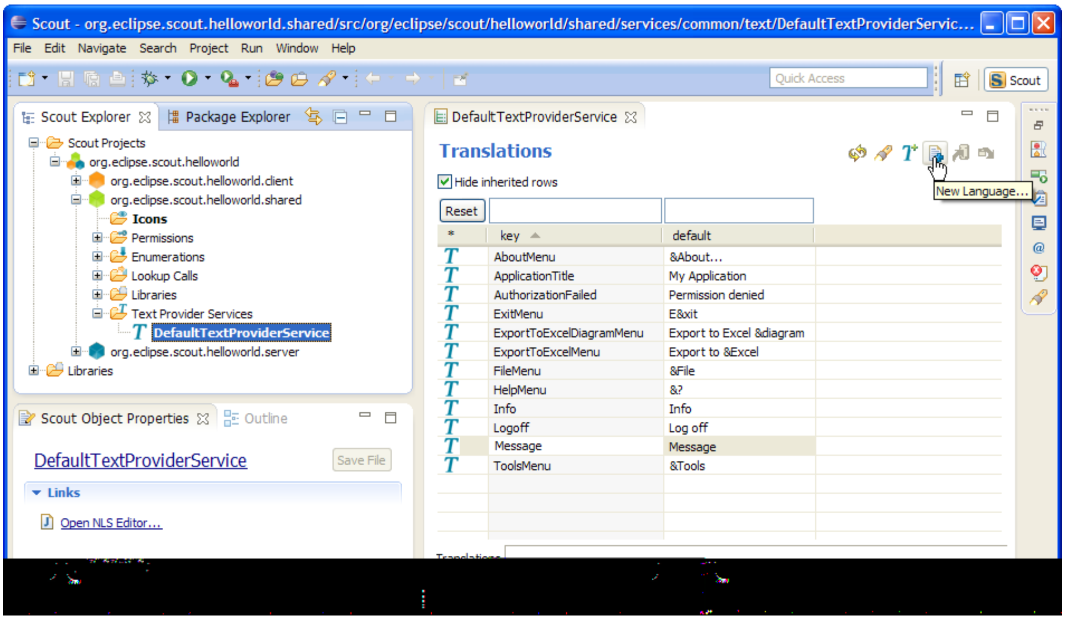
Sibling: last

MessageField [before]

last

2 items found

Figure 4.15: Adding a text translation for a field label (left) and specifying the ordering of the fields (right).


Scout - org.eclipse.scout.helloworld.shared/src/org/eclipse/scout/helloworld/shared/services/common/text/DefaultTextProviderServic…
File Edit Navigate Search Project Run Window Help
Quick Access
Scout
Scout Explorer
Package Explorer
Scout Projects
org.eclipse.scout.helloworld
org.eclipse.scout.helloworld.client
org.eclipse.scout.helloworld.shared
Icons
Permissions
Enumerations
Lookup Calls
Libraries
Text Provider Services
DefaultTextProviderService
org.eclipse.scout.helloworld.server
Libraries
Scout Object Properties
Outline
DefaultTextProviderService
Save File
Links
Open NLS Editor…
DefaultTextProviderService
New Language…

Translations

☑ Hide inherited rows

Reset

*	key	default
T	AboutMenu	&About…
T	ApplicationTitle	My Application
T	AuthorizationFailed	Permission denied
T	ExitMenu	E&xit
T	ExportToExcelDiagramMenu	Export to Excel &diagram
T	ExportToExcelMenu	Export to &Excel
T	FileMenu	&File
T	HelpMenu	&?
T	Info	Info
T	Logoff	Log off
T	Message	Message
T	ToolsMenu	&Tools

Figure 4.16: The NLS editor provided by the Scout SDK. This editor is opened via the *Open NLS Editor …* link in the Scout Object Properties of the *DefaultTextProviderService* node.

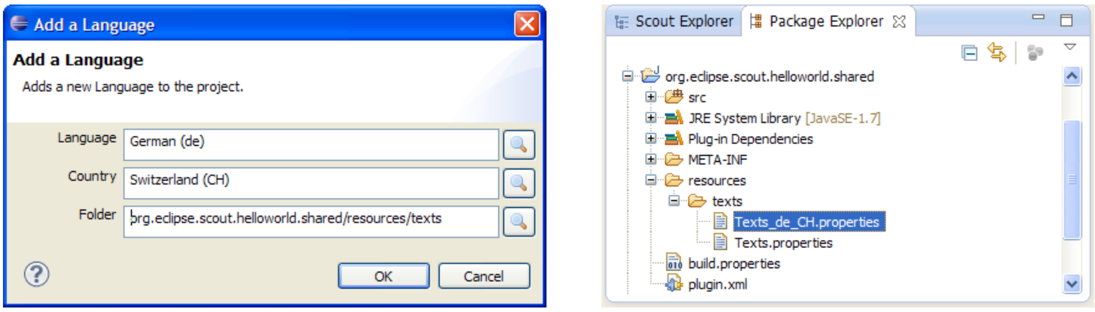
shown on the right hand side of Figure 4.15, the new `PreferencesField` is to be placed before the message field.

4.5.4 The NLS Editor

Access to translated texts in Scout applications is provided through text provider services located in the application's shared plugin. This setup makes translated texts available in the client and the server application as the shared plugin is available in both the client and the server product. Consequently, the Scout SDK provides access to the NLS Editor to manage translated texts and application languages under the green shared node in the Scout Explorer view as shown in Figure 4.16.

To open the NLS editor, select the green *shared* node in the Scout Explorer and expand its *Text Provider Services* folder. Then, select the contained element *DefaultTextProviderService*. As shown in Figure 4.16, the NLS editor can be opened using the *Open NLS Editor …* link. In the NLS editor translations can be edited individually by double-clicking into a translated text cell. This opens the editor to change the text to the desired value. To add a new translated text, the corresponding dialog can be opened by clicking on the button with the 'T' icon next to the 'New language' button (shown in Figure 4.16).

In Scout applications, translated texts are obtained with the method `TEXTS.get("key")` where "key" represents the language independent text key. In a default Scout project setup, calling `TEXTS.get` uses the `DefaultTextProviderService` in the background. And this text provider service defines the access path for the text property files that are located in the applications shared plugin. Typically, these property files are collected in the plugin's `<youraplication>.shared/resources/texts` directory according to the right hand side of Figure 4.17. To



Add a Language

Adds a new Language to the project.

Language: German (de)

Country: Switzerland (CH)

Folder: org.eclipse.scout.helloworld.shared/resources/texts

Figure 4.17: In the dialog *Add a Language* shown on the left side the desired language and localization can be specified. On the right side, the location of property file `Texts_de_CH.properties` in the shared plugin of the "Hello World" application is shown.

resolve the provided text key at runtime, the user's locale settings is used to access the correct text property file.

Adding support for a new language to a Scout project is also directly supported by the Scout SDK. To add support for a new translated language, click on the corresponding icon of the NLS editor as shown in Figure 4.16. In the opened language dialog, add the desired language and country localization and the target directory as shown in Figure 4.17. A new language properties file is then added to the shared plugin project as shown in Figure 4.17. The new language is now available as an additional column in the NLS editor and as a separate tab in the dialog to add new translated texts for Scout UI components such as form field labels.

Chapter 5

A Larger Example

In this chapter we will create the "My Contacts" Scout application. This small[1] application covers additional aspects of the Eclipse Scout framework. The presented demo application borrows heavily from a Scout tutorial published 2012 in the German *Java Magazin*[2] for the Scout release 3.8 (Juno). Compared to the 2012 tutorial, the version presented in this chapter has been slightly polished and updated to the Scout release 3.9 (Kepler).

Specifically, we will build an outline based Scout application featuring a navigation tree and pages to present information in tabular form. In addition, the application also shows how to work with forms to enter and/or update data, menus and context menus. On the server side, we show how to work with databases, how to use logging in Scout applications and how to include standard Java libraries available in JAR files.

The chapter is organized as follows. In the first section, the finished demo application is explained from the user perspective. The remaining sections focus on the individual steps to implement the "My Contacts" application. To follow the description of the implementation the reader is assumed to be familiar with the "Hello World" tutorial and the Scout SDK as described in Chapter 4.

5.1 The "My Contacts" Application

The "My Contacts" application is a client server application to manage personal contacts. Persistence of entered data is achieved through a database backend.

As social networking services[3] such as Facebook, LinkedIn or Xing are widely used, the application also provides an example integration with the LinkedIn[4] platform. The implemented integration allows to download the personal contacts into the local database. In the local database it is possible to mix persons entered manually with contact data downloaded from LinkedIn.

After starting the Scout server application a client application may be started that then

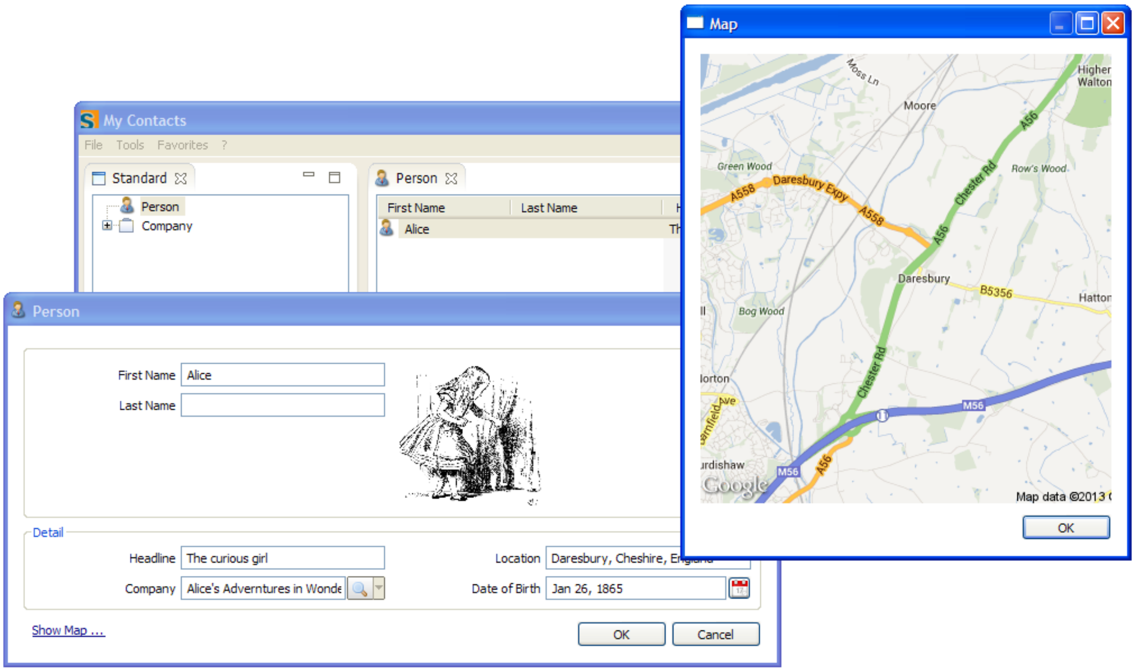
---

[1] Small in comparison with real world application. But significantly larger and complexer than the "Hello World" application of Chapter 2.

[2] Java Magazin 7.12: `https://jaxenter.de/magazines/JavaMagazin72012`.

[3] Social networking services in Wikipedia: `http://en.wikipedia.org/wiki/Social_network_service`

[4] `http://www.linkedin.com/`



First Name	Last Name
Alice	

First Name: Alice

Last Name:

Headline: The curious girl

Company: Alice's Adverntures in Wonde

Location: Daresbury, Cheshire, England

Date of Birth: Jan 26, 1865

Figure 5.1: The SWT client of the "My Contacts" application.

connects to the server. In Figure 5.1 the SWT desktop client is shown. In the background, the main application window is visible showing a navigation tree on the left hand side. On the right side, a table holds the elements corresponding to the selected tree node. Using an edit context menu on the selected table row, a form to edit the relevant data may be opened as shown in the example screen shot for 'Alice'. Clicking on the link 'Show Map ...' in the person form opens the person's location information in a map form using the data provided by Google Maps Image API[5].

Before any LinkedIn data can be accessed from the "My Contacts" application an access token needs to be retrieved from LinkedIn. To obtain such a token use the REFRESH LINKEDIN TOKEN ... menu as shown in Figure 5.2. This opens the refresh token form shown on the left side of Figure 5.3.

Clicking on the 'Open Auth URL' link then opens the granting page provided by LinkedIn shown on the left hand side of Figure 5.4. After logging into your LinkedIn account[6] you can specify the desired access duration and confirm the "OAuth test for Scribe" access[7] to your LinkedIn data. If the authorization is successful, a security code as shown on the right side of Figure 5.4 is presented by LinkedIn. This code needs then to be entered into the *Security Code* field as shown on the right side of Figure 5.3. Then, click the OK button to refresh the access token.

To import/update your LinkedIn contacts into your "My Contacts" application select the UPDATE LINKEDIN CONTACTS menu as shown in Figure 5.5. Once you have downloaded or entered a number of persons in your "My Contacts" application, try to get yourself familiar with

---

[5] The Google Maps Image APIs: `https://developers.google.com/maps/documentation/imageapis/`.

[6] Yes, for this use case you need a LinkedIn account. But at least it's free and you will not need to provide very sensitive information such as a mobile phone number, a credit card number or a social security number.

[7] This is the name of the example code provided with the Scribe library that is used with the "My Contacts" application.

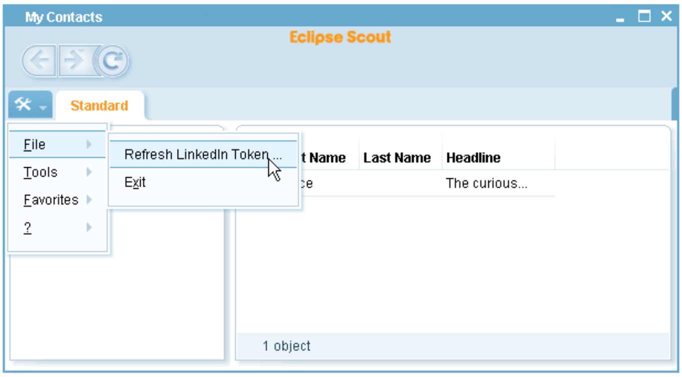
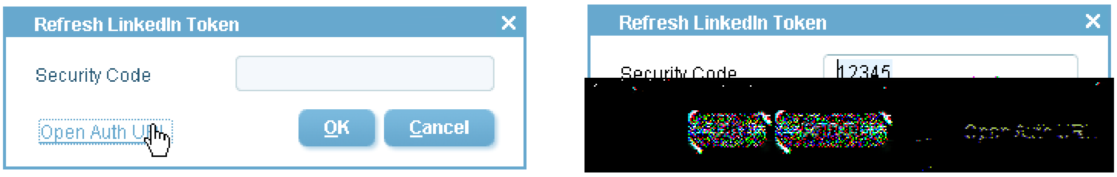
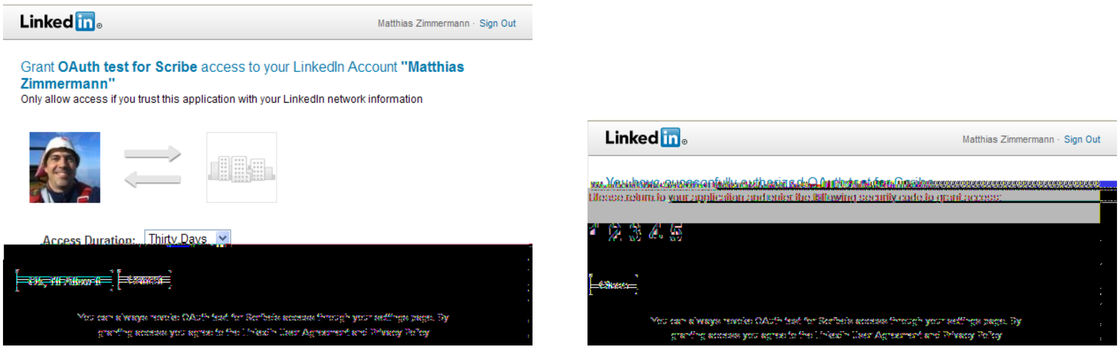


Figure 5.2: To refresh/generate an access token for reading LinkedIn contacts data select the menu shown in the screen shot.



Security Code 12345

Figure 5.3: To refresh/create the access token click on the provided 'Open Auth URL' link first (left). Then, the security code field is enabled and the user can fill in the security code provided on the LinkedIn web page.


Linked in
Matthias Zimmermann · Sign Out
Access Duration: Thirty Days
Linked in
Matthias Zimmermann · Sign Out

Grant **OAuth test for Scribe** access to your LinkedIn Account **"Matthias Zimmermann"**

Only allow access if you trust this application with your LinkedIn network information

Please return to your application and enter the following security code to grant access:

12345

Figure 5.4: The LinkedIn granting dialog steps shown in a web browser. In the first step (right) confirm the access request, then the security code to create an access token is provided in the second step (left).

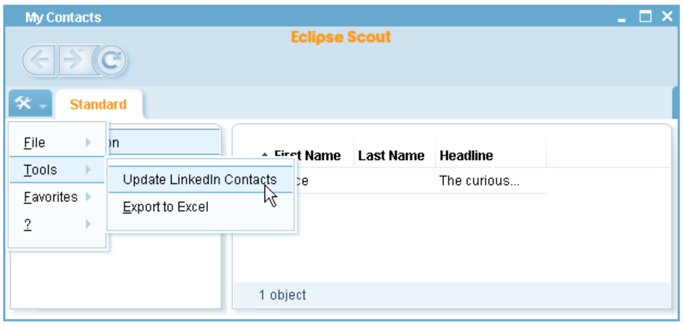


Figure 5.5: Executing the 'Update LinkedIn Contacts for the first time imports the users LinkedIn contacts into the "My Contacts" application.

the application's person table. This is one of the very powerful Scout widgets. Columns may be filtered, moved, hidden or sorted (including multi level sort) using the table header context menus ORGANIZE COLUMNS... menu and COLUMN FILTER... menu.

Editing and viewing of person data is available by the EDIT PERSON... context menu on a selected row. To manually add a person use the NEW PERSON... context menu available on the table header or in the white area outside the displayed columns.

In Figure 5.7 the "My Contacts" application is running in a web browser. In this example, the TOOLS—EXPORT TO EXCEL menu is used to export the selected row into an Excel sheet. Finally, the "My Contacts" application is also running on iPhone and Android mobile devices out of the box. Two example screens are provided in Figure 5.8.

Once you no longer feel confident about having the "My Contacts" application accessing you data you can revoke this access permission in the LinkedIn menu "Privacy and Settings". In the lower part of the settings page switch to the tab "Groups, Companies and Applications" and click on the link "View your applications". There, you will find again the partner name "OAuth test for Scribe". To revoke the access, select the associated checkbox and click the "Remove" button. The next time you try to refresh your LinkedIn data from the "My Contacts" application will result in an Error message. Before you can again access data from your LinkedIn account you just need to refresh the access token as described above.

To run the "My Contacts" Scout application without implementing it first, you may take advantage of the fact that the application is hosted in the same Github repository as this book. If you are familiar with Github[8], fork the Scout Book repository[9] and start from there. Alternatively, you can follow the description provided in the Scout wiki[10] to download, install and run the "My Contacts" application.

5.2 Setting up the new Scout project

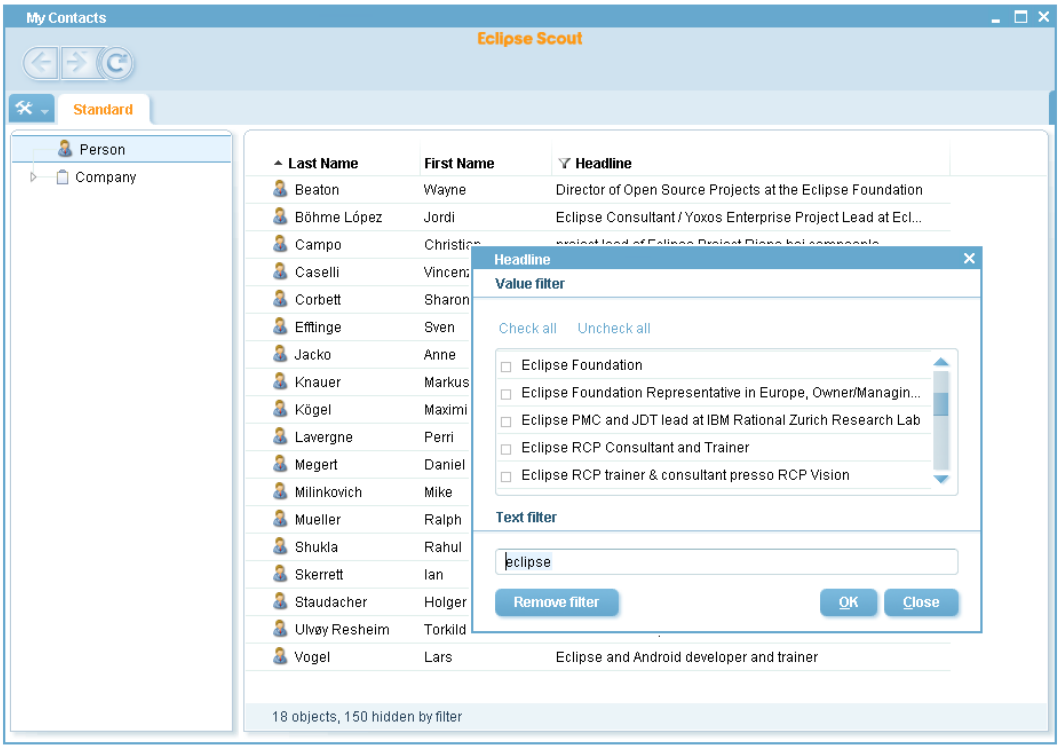
The initial code for the "My Contacts" application is generated using the *New Scout Project* wizard as described in Section 4.5.1. For the *Project Name* field use the name

---

[8] Github: https://en.wikipedia.org/wiki/GitHub.

[9] Scout Book repository: https://github.com/BSI-Business-Systems-Integration-AG/scoutbook.

[10] Download and installation of the "My Contacts" application: http://wiki.eclipse.org/Scout/Book/3.9#Download_and_Run_the_Scout_Sample_Applications.



Last Name	First Name	Headline
Beaton	Wayne	Director of Open Source Projects at the Eclipse Foundation
Böhme López	Jordi	Eclipse Consultant / Yoxos Enterprise Project Lead at Ecl...
Campo	Christia	
Caselli	Vincen	
Corbett	Sharon	
Efftinge	Sven	
Jacko	Anne	
Knauer	Markus	
Kögel	Maximi	
Lavergne	Perri	
Megert	Daniel	
Milinkovich	Mike	
Mueller	Ralph	
Shukla	Rahul	
Skerrett	Ian	
Staudacher	Holger	
Ulvøy Resheim	Torkild	
Vogel	Lars	Eclipse and Android developer and trainer

18 objects, 150 hidden by filter

Headline

Value filter

Check all Uncheck all

☐ Eclipse Foundation
☐ Eclipse Foundation Representative in Europe, Owner/Managin...
☐ Eclipse PMC and JDT lead at IBM Rational Zurich Research Lab
☐ Eclipse RCP Consultant and Trainer
☐ Eclipse RCP trainer & consultant presso RCP Vision

Text filter: eclipse

Figure 5.6: After importing the contacts from LinkedIn the data is shown in the person page. The filter applied on the headline column is indicated by the filter icon. In the front, the filter form shows the filter criteria 'Eclipse'.

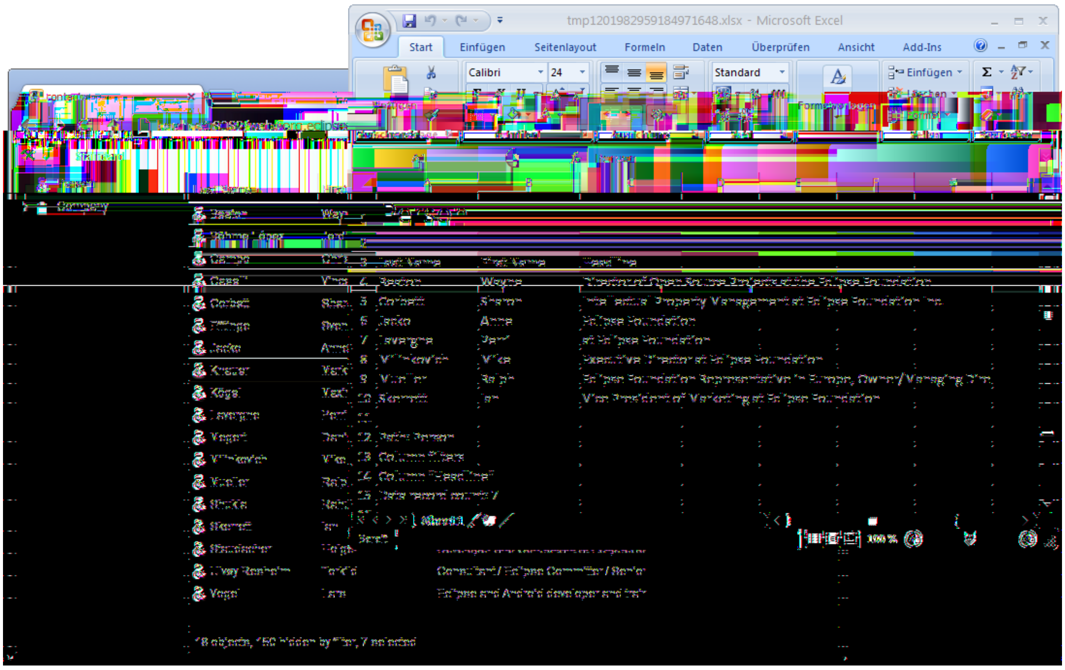


Figure 5.7: The "My Contacts" application running in the browser as web application. The Excel sheet shown in the front is exported from the person page using the 'Tools/Export to Excel' menu.

`org.eclipsescout.contacts` as shown on the left side of Figure 5.9 and click on the NEXT button. In the second wizard step select the application template *Outline Tree and Table Form* as shown on the right side of Figure 5.9.

After the Scout SDK has created the initial application code select the top-level *org.eclipsescout.contacts* node in the Scout Explorer. In the technology section of the corresponding Scout Object Properties select the Derby database driver, the Docx4j support and the Rayo look and feel as shown on the right side of Figure 5.10. In case you have not yet used the Scout Docx4j support or the Rayo look and feel components, the Scout SDK will need to download these packages from the Eclipse Marketplace[11] first.

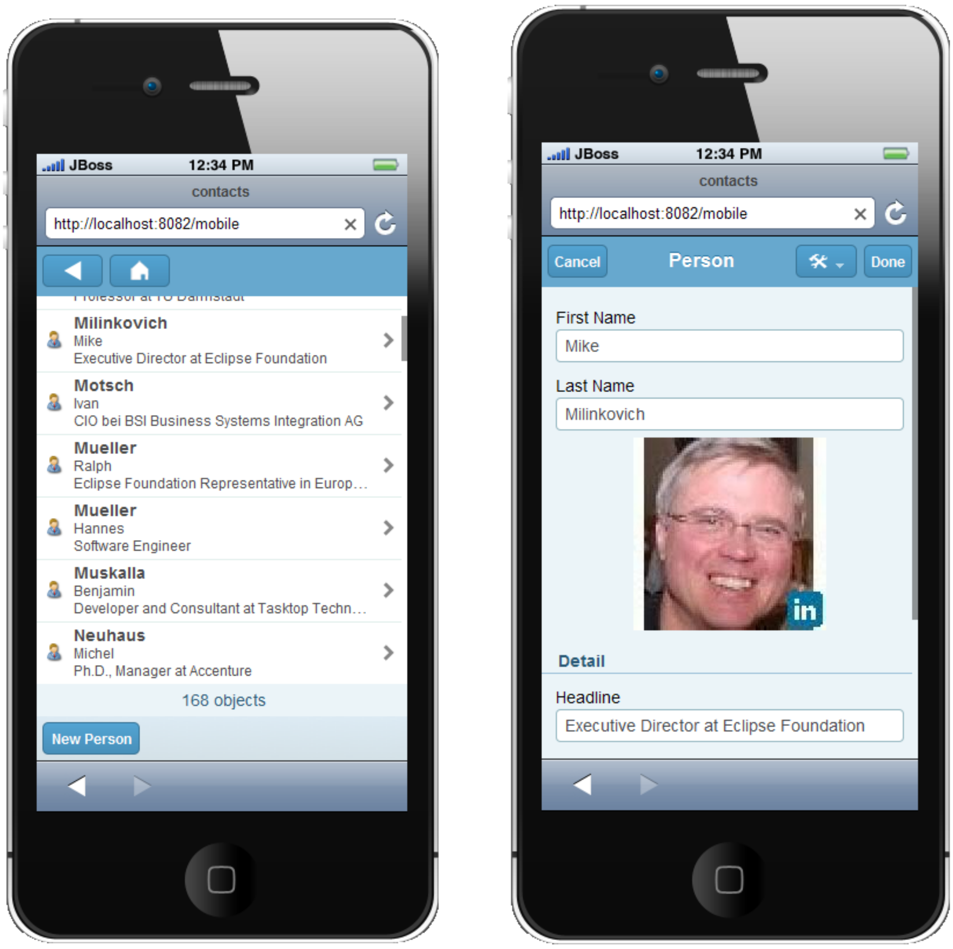
Adding the Docx4j support will also add the EXPORT TO EXCEL menu under tools menu on the application's desktop. As shown in Listing 5.1, the `ScoutXlsxSpreadsheetAdapter` first creates an excel file based on the currently active page in the `execAction` method. Then the file is opened on the client using the `shellOpen` method.

To set the applications name select the *Desktop* node in the Scout Explorer to access it's Scout Object Properties as shown in Figure 5.11. In the *Title* field enter the string "My Contacts" and create a new translated text entry.

As shown in Listing 5.2 the application's organisation into a tree and a table form is explicitly defined in method `execOpened`. Click on the *Exec Opened* link in the Scout Object Properties to access the Java code of this method. First, using the `UserAgentUtility` class, the method checks if the client is working with a desktop or a web client. If not, the method returns and the tree and table setup is not used. Instead, the `MobileHomeForm` defined in plugin

---

[11] See Section 4.4 for additional information regarding the download of marketplace packages



Milinkovich
Mike
Executive Director at Eclipse Foundation

Motsch
Ivan
CIO bei BSI Business Systems Integration AG

Mueller
Ralph
Eclipse Foundation Representative in Europ...

Mueller
Hannes
Software Engineer

Muskalla
Benjamin
Developer and Consultant at Tasktop Techn...

Neuhaus
Michel
Ph.D., Manager at Accenture

168 objects

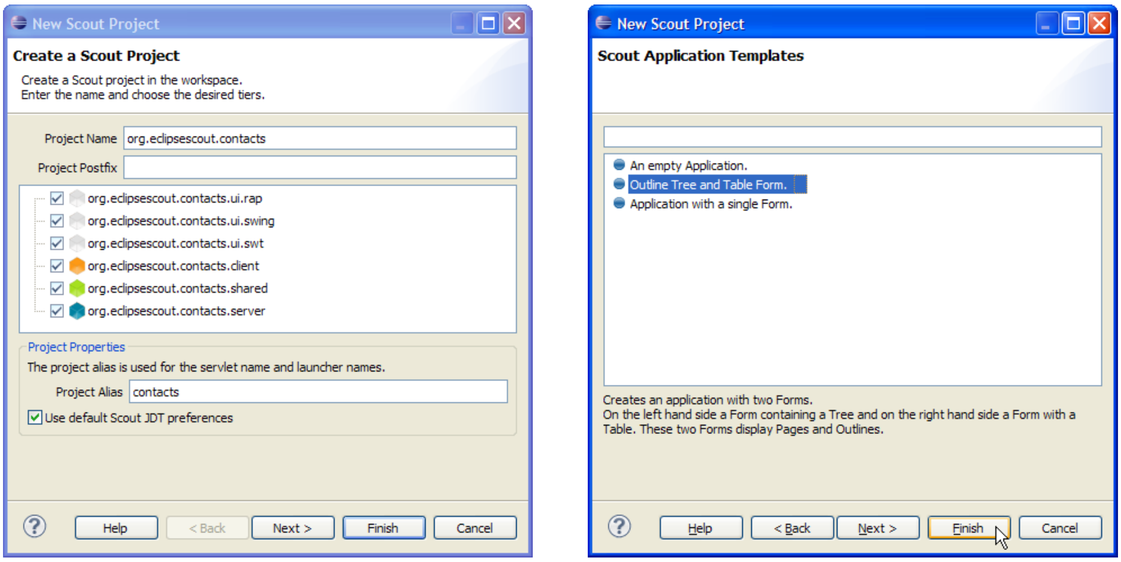
First Name: Mike

Last Name: Milinkovich

Detail

Headline: Executive Director at Eclipse Foundation

Figure 5.8: The "My Contacts" application running on an iPhone device. On the left hand side, the person page is shown. The person form is shown on the right.



Create a Scout Project

Create a Scout project in the workspace.
Enter the name and choose the desired tiers.

Project Name: org.eclipsescout.contacts
Project Postfix:

- ☑ org.eclipsescout.contacts.ui.rap
- ☑ org.eclipsescout.contacts.ui.swing
- ☑ org.eclipsescout.contacts.ui.swt
- ☑ org.eclipsescout.contacts.client
- ☑ org.eclipsescout.contacts.shared
- ☑ org.eclipsescout.contacts.server

Project Properties

The project alias is used for the servlet name and launcher names.

Project Alias: contacts

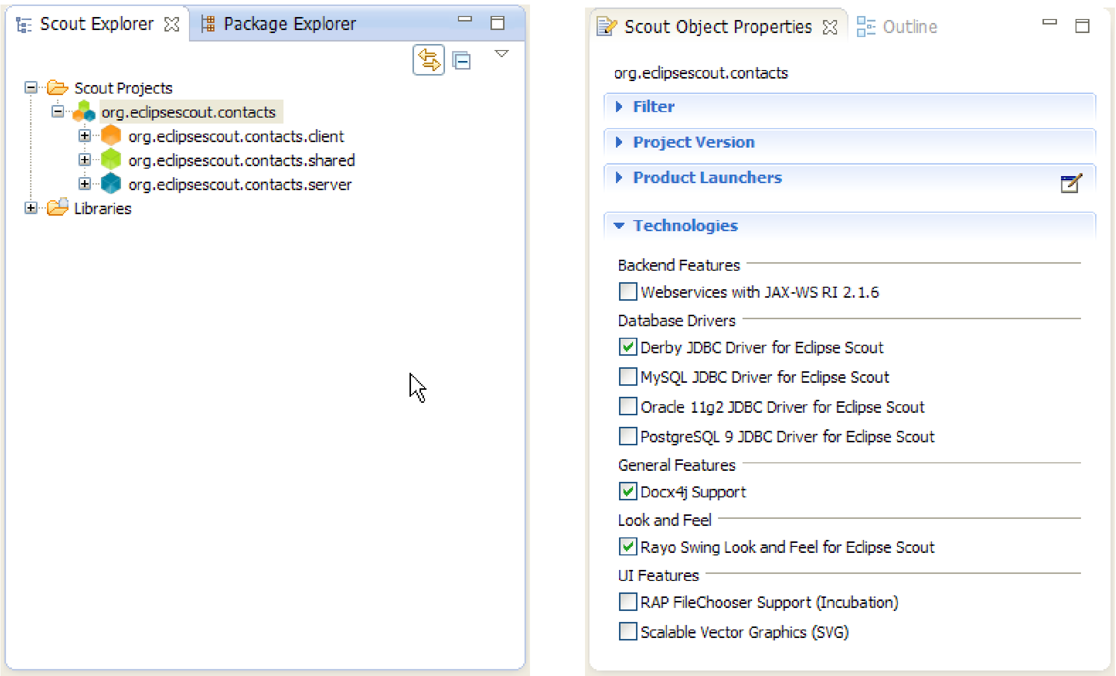
☑ Use default Scout JDT preferences

Scout Application Templates

- An empty Application.
- Outline Tree and Table Form.
- Application with a single Form.

Creates an application with two Forms.
On the left hand side a Form containing a Tree and on the right hand side a Form with a Table. These two Forms display Pages and Outlines.

Figure 5.9: Start with creating a new Scout project.



- Scout Projects
  - org.eclipsescout.contacts
    - org.eclipsescout.contacts.client
    - org.eclipsescout.contacts.shared
    - org.eclipsescout.contacts.server
- Libraries

org.eclipsescout.contacts

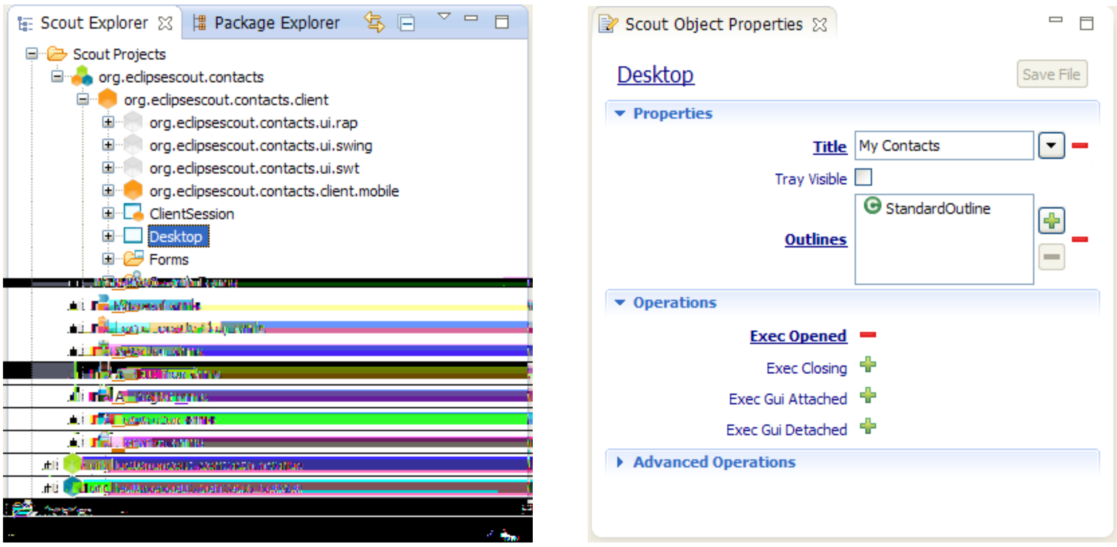
- Filter
- Project Version
- Product Launchers
- Technologies

Backend Features
☐ Webservices with JAX-WS RI 2.1.6
Database Drivers
☑ Derby JDBC Driver for Eclipse Scout
☐ MySQL JDBC Driver for Eclipse Scout
☐ Oracle 11g2 JDBC Driver for Eclipse Scout
☐ PostgreSQL 9 JDBC Driver for Eclipse Scout
General Features
☑ Docx4j Support
Look and Feel
☑ Rayo Swing Look and Feel for Eclipse Scout
UI Features
☐ RAP FileChooser Support (Incubation)
☐ Scalable Vector Graphics (SVG)

Figure 5.10: The setting of the *Technology* section in the Scout Object Properties of the ”My Contacts” application.

Listing 5.1: The `ExportToExcelMenu` class added by the Docx4j support to the application's tools menu.

```
 @Order(20.0)
 public class ExportToExcelMenu extends AbstractExtensibleMenu {

   @Override
   protected String getConfiguredText() {
     return TEXTS.get("ExportToExcelMenu");
   }

   @Override
   protected void execAction() throws ProcessingException {
     if (getOutline() != null && getOutline().getActivePage() != null) {
       ScoutXlsxSpreadsheetAdapter s = new ScoutXlsxSpreadsheetAdapter();
       File xlsx = s.exportPage(null, 0, 0, getOutline().getActivePage());
       SERVICES.getService(IShellService.class).shellOpen(xlsx.↘
         →getAbsolutePath());
     }
   }
  }
}
```



Title: My Contacts
Tray Visible ☐
Outlines: StandardOutline

Figure 5.11: Configure the application name in the title field of the Desktop's properties.

Listing 5.2: The `execOpened` method of desktop class of the ”My Contacts” application. The application’s organisation into a tree and a table form is defined here.

```
@Override
protected void execOpened() throws ProcessingException {
  //a mobile home form (in client.mobile plugin) is used instead.
  if (!UserAgentUtility.isDesktopDevice()) {
    return;
  }

  // outline tree
  DefaultOutlineTreeForm treeForm = new DefaultOutlineTreeForm();
  treeForm.setIconId(Icons.EclipseScout);
  treeForm.startView();

  //outline table
  DefaultOutlineTableForm tableForm = new DefaultOutlineTableForm();
  tableForm.setIconId(Icons.EclipseScout);
  tableForm.startView();

  if (getAvailableOutlines().length > 0) {
    setOutline(getAvailableOutlines()[0]);
  }
}
```

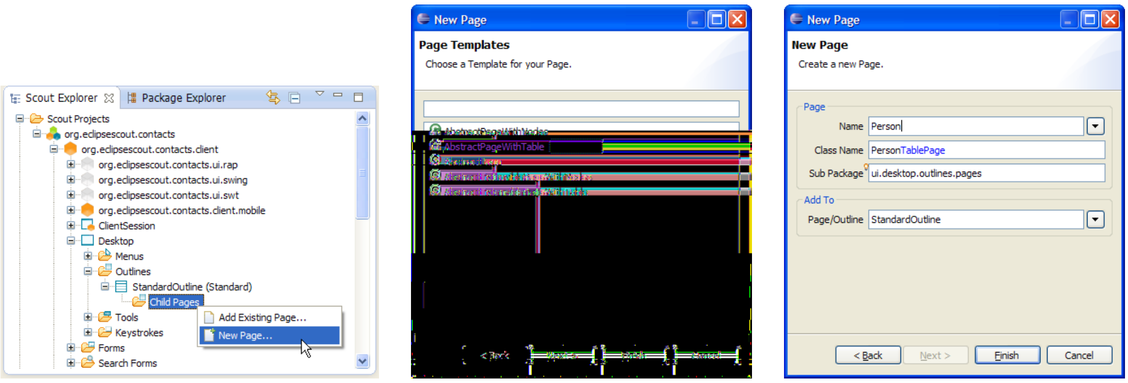
org.eclipsescout.contacts.client.mobile is used. In case the user is working with a web client or a desktop client, default tree and table forms are created and started. Finally, the currently active outline is set to the `StandardOutline`, as this is the only outline defined in this application.

5.3 Adding the Person Page

The first UI component we add to the application is the person page. For the desktop clients, this page is represented as a table that will list all available persons in the database of the ”My Contacts” application. To start the *New Page* wizard use the NEW PAGE... context menu on the *Child Pages* folder as shown in Figure 5.12. On the first wizard step select the template *AbstractPageWithTable* and click the NEXT button. On the second wizard step, provide the page name ”Person” to create a new translated text entry at the same time. Make sure the other fields are filled in as shown in Figure 5.12 and click the FINISH button to close the wizard.

Listing 5.3 shows the created method `execCreateChildPages` that links the newly created person page to the standard outline. Note that your code will only look the same once you have added the company page in a later step of the implementation of this application.

Now, drill down to the *Columns* folder under the *PersonTablePage* node as shown in Figure 5.14. Here we can add the desired table columns to the person page. Start with the column that will hold the person id. For this, start the column wizard as shown on the left side of Figure 5.14 and select the string column template. In the second wizard step, enter ”PersonId” into the *Class Name* field,select the radio button *Create one more column* and click on the FINISH button. This will restart the column wizard to enter the next columns. Create the remaining string columns with the following names.

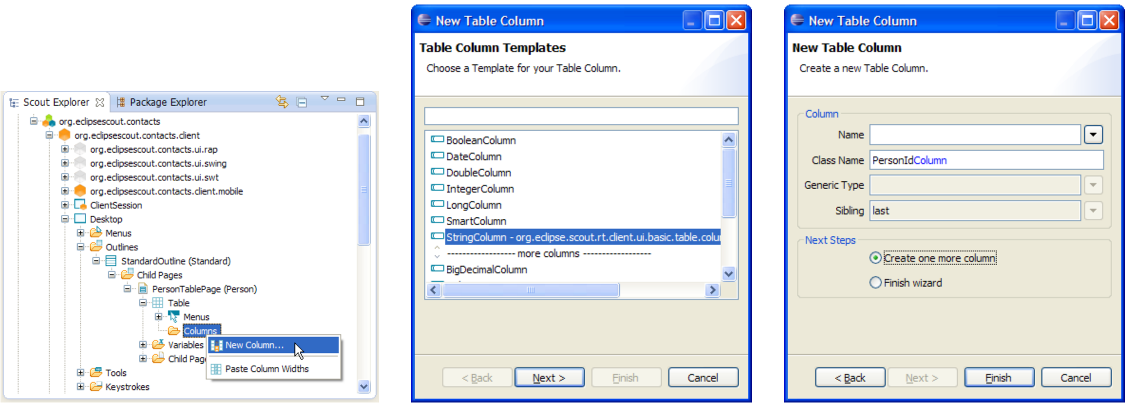


Name: Person
Class Name: PersonTablePage
Sub Package: ui.desktop.outlines.pages
Page/Outline: StandardOutline

Figure 5.12: Add the person table page below the standard outline.

Listing 5.3: The `execCreateChildPages` method of the standard outline. At the current implementation step the company table page is not (yet) added.

```
 @Override
 protected void execCreateChildPages(Collection<IPage> pageList) throws ↘
   →ProcessingException {
  PersonTablePage personTablePage = new PersonTablePage();
  pageList.add(personTablePage);
  pageList.add(new CompanyTablePage());
 }
}
```



Name:
Class Name: PersonIdColumn
Generic Type:
Sibling: last
● Create one more column
○ Finish wizard

Figure 5.13: Add columns to the person page.

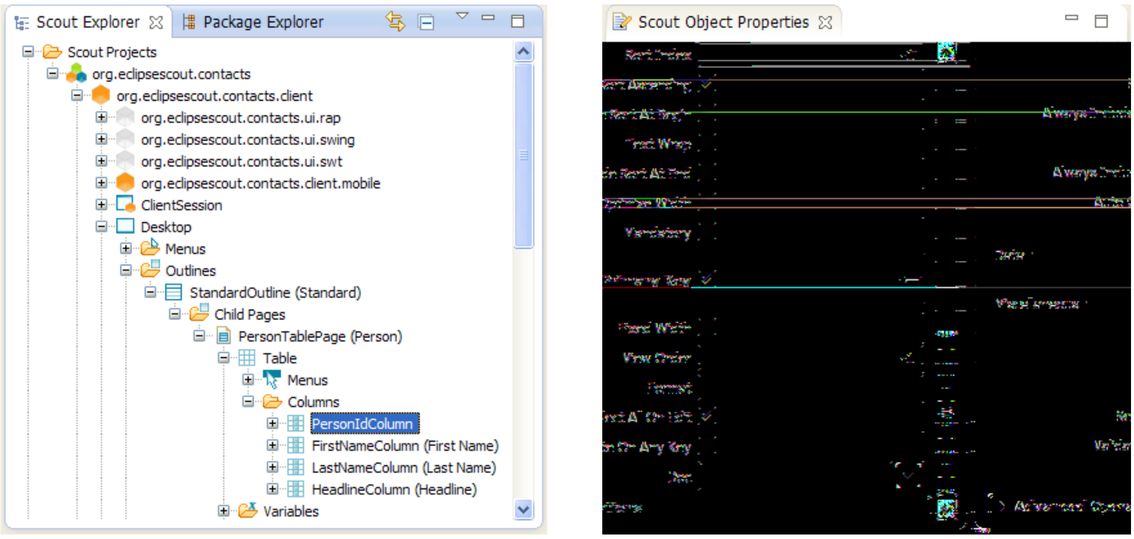
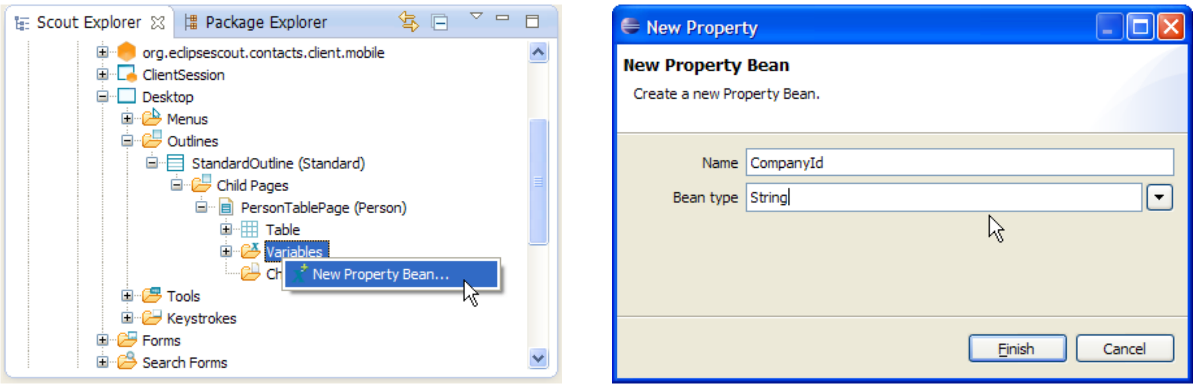

Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.contacts
org.eclipsescout.contacts.client
org.eclipsescout.contacts.ui.rap
org.eclipsescout.contacts.ui.swing
org.eclipsescout.contacts.ui.swt
org.eclipsescout.contacts.client.mobile
ClientSession
Desktop
Menus
Outlines
StandardOutline (Standard)
Child Pages
PersonTablePage (Person)
Table
Menus
Columns
PersonIdColumn
FirstNameColumn (First Name)
LastNameColumn (Last Name)
HeadlineColumn (Headline)
Variables
Scout Object Properties

Figure 5.14: Configure the *PersonId* column. Check property *Primary Key* under the section *Advanced Properties* (right).



Name: CompanyId
Bean type: String

Figure 5.15: Add the *CompanyId* variable to the person page.

- First Name
- Last Name
- Headline

Once you have created all these columns we will mark the person id column as the primary key column for the person page. In the Scout Explorer select the *PersonIdColumn* node to open the corresponding Scout Object Properties. In this form, deselect the *Displayable* property to always hide this technical column from the end user. In the properties *Advanced Properties* section check the *Primary Key* property. The resulting Java code for the primary key column and the first name column is provided in Listing 5.5.

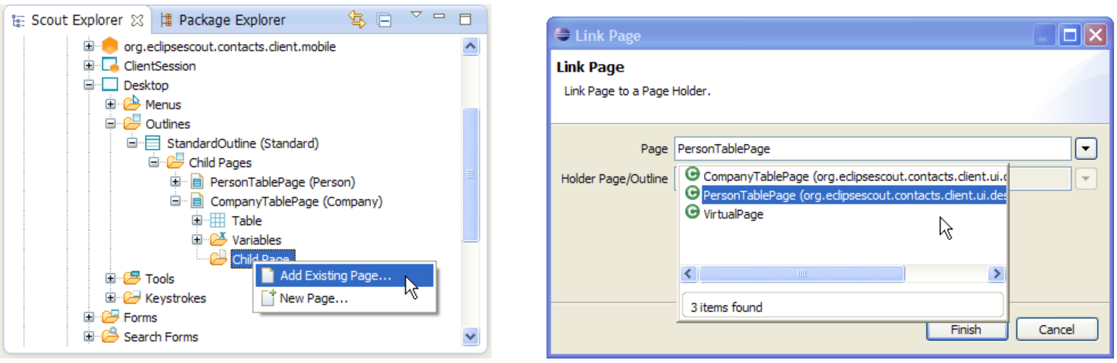
As we will later add and link the person page with a company page, we add a company id variable to the person page as shown in Figure 5.15. For the Java representation of such variables the standard bean pattern is used as shown in Listing 5.5.

Listing 5.4: The person id and the first name columns of the `PersonTablePage` class.

```
    @Order(10.0)
    public class PersonIdColumn extends AbstractStringColumn {

      @Override
      protected boolean getConfiguredDisplayable() {
        return false;
      }

      @Override
      protected boolean getConfiguredPrimaryKey() {
        return true;
      }
    }

    @Order(20.0)
    public class FirstNameColumn extends AbstractStringColumn {

      @Override
      protected String getConfiguredHeaderText() {
        return TEXTS.get("FirstName");
      }
    }
```

Listing 5.5: The company id variable of the `PersonTablePage` class.

```
public class PersonTablePage extends AbstractPageWithTable<PersonTablePage.↘
  →Table> {

  private String m_companyId;
  @FormData
  public String getCompanyId() {
    return m_companyId;
  }

  @FormData
  public void setCompanyId(String companyId) {
    m_companyId = companyId;
  }
}
```



Link Page

Link Page to a Page Holder.

Page PersonTablePage

Holder Page/Outline

CompanyTablePage (org.eclipsescout.contacts.client.ui.c
PersonTablePage (org.eclipsescout.contacts.client.ui.des
VirtualPage

3 items found

Figure 5.16: Add the person page below the the company page.

Listing 5.6: Linking the `PersonTablePage` with the parent `CompanyTablePage`.

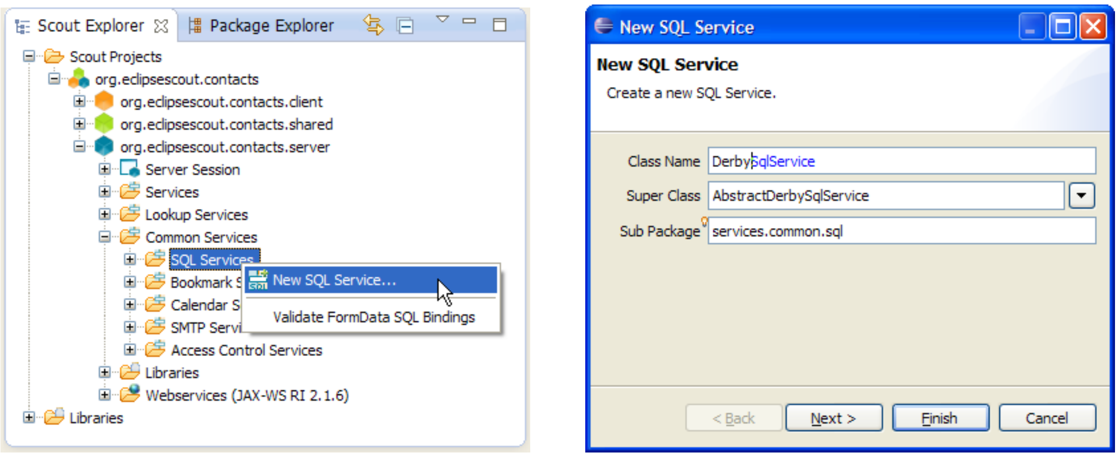
```
@Override
protected IPage execCreateChildPage(ITableRow row) throws ↘
  →ProcessingException {
  PersonTablePage childPage = new PersonTablePage();
  childPage.setCompanyId(getTable().getCompanyIdColumn().getValue(row));
  return childPage;
}
```

5.4 Adding the Company Page

After adding the person page, we add a company page to the standard outline. To add the company page we use the same wizards as described in the previous section for the person page. For the *Name* field we enter the new translated text "Company" and the columns to add are the following.

- Company Id
- Name
- Location

As in the case of the person table page, the company id column is used as a primary key column. The *Displayable* property needs to be set to false and the *Primary Key* property to true. Now we can link the person page to the company page using the ADD EXISTING PAGE... context menu as shown on the left side of Figure 5.16. In the *Link Page* wizard, the person page can then be selected according to the right side of Figure 5.16. In the Java code generated by the Scout SDK the setting of the company id attribute is automatically inserted in method `execCreateChildPages`. Please note that this convenience added by the Scout SDK wizard only works if the child page defines variables with a naming that matches the defined primary key columns of the parent table.



New SQL Service

Create a new SQL Service.

Class Name: DerbySqlService

Super Class: AbstractDerbySqlService

Sub Package: services.common.sql

Figure 5.17: Add the service to access the Derby database under folder *SQL Services*.

Listing 5.7: Setting up the database parameters in the Scout server's `config.ini` file.

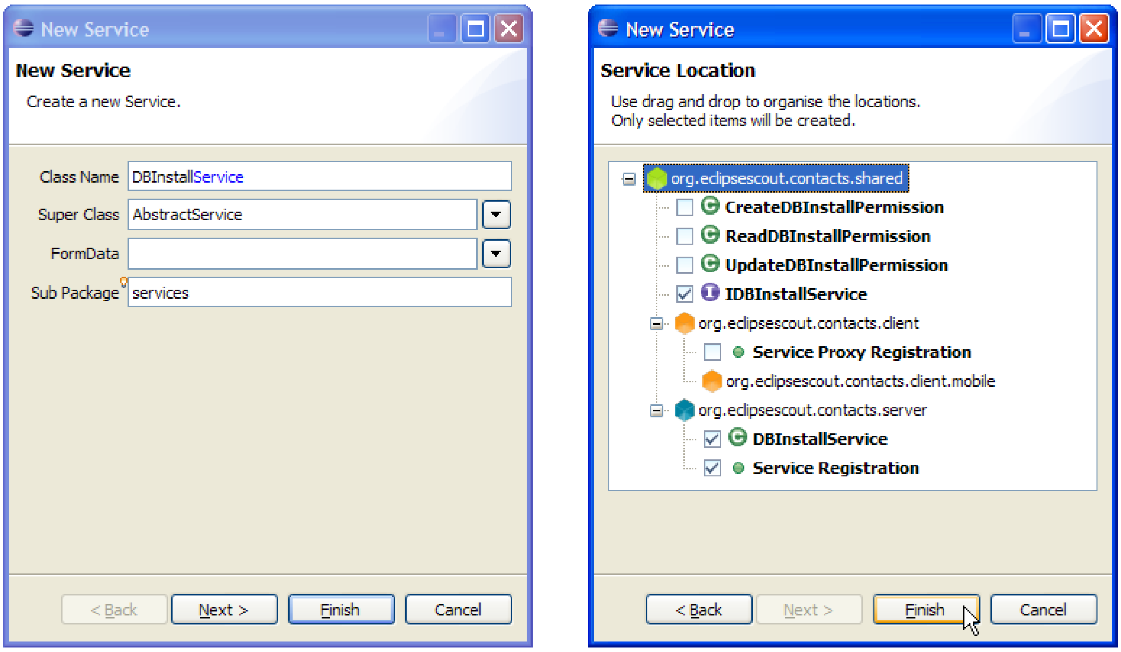
```
### DataSource
org.eclipsescout.contacts.server.services.common.sql.DerbySqlService#↘
  →directJdbcConnection=true
org.eclipsescout.contacts.server.services.common.sql.DerbySqlService#↘
  →jdbcDriverName=org.apache.derby.jdbc.EmbeddedDriver
org.eclipsescout.contacts.server.services.common.sql.DerbySqlService#↘
  →jdbcMappingName=jdbc:derby:${workspace_loc}/../DB_CONTACT;create=true;↘
  →territory=en_US
org.eclipsescout.contacts.server.services.common.sql.DerbySqlService#username↘
  →=contact_user
org.eclipsescout.contacts.server.services.common.sql.DerbySqlService#password↘
  →=secr3t
```

5.5 Installing the Database

To access a database we first need to install a database service. For the "My Contacts" application, this is done using the *New SQL Service* wizard on the Scout server under the *SQL Services* folder as shown in Figure 5.17. In the first wizard step shown on the right side of the figure, use "DerbySqlService" for the *Class Name* field. From the drop-down list in the *Super Class* field choose the `AbstractDerbySqlService` and click on the FINISH button.

To setup the database connection the necessary parameters need to be added to the server's `config.ini` file as shown in Listing 5.7. Comparing the parameter names in this config file with the package name of the created `DerbySqlService` service class reveals an interesting framework feature. All Scout services can be parameterized using the `config.ini` file with the pattern `<package>.<class name>#<parameter>=<value>`. The Scout runtime then sets the service parameters using matching setter methods such as `setPassword` for the password parameter.

According to the parameter `jdbcMappingName=jdbc:derby:${workspace_loc}/...` a new Derby database will be created in the same parent directory as the "My Contacts" workspace



Class Name DBInstallService
Super Class AbstractService
FormData
Sub Package services

- org.eclipsescout.contacts.shared
  - ☐ CreateDBInstallPermission
  - ☐ ReadDBInstallPermission
  - ☐ UpdateDBInstallPermission
  - ☑ IDBInstallService
  - org.eclipsescout.contacts.client
    - ☐ Service Proxy Registration
    - org.eclipsescout.contacts.client.mobile
  - org.eclipsescout.contacts.server
    - ☑ DBInstallService
    - ☑ Service Registration

Figure 5.18: Add a database installation service.

directory if no database named `DB_CONTACT` is found there. This setup is handy for development purposes but you may want to set the database parameter to `create=false` in the `config.ini` of the production product file.

5.5.1 Setting up the Database Schema

Having a working database service and a new (empty) Derby database now allows to create the necessary database schema and populate it with some initial data. For this we add a new `DBInstallService` service as shown in Figure 5.18. To this install service we then add the `installStorage` operation according to Figure 5.19. The implementation of this method is provided in Listing 5.8. The method first checks if a setup is required. For this, the member variable `m_doSetup` is used that might also be set by the `setDoSetup` setter method via the server's `config.ini` file.

Setting up the schema to contain the individual tables for the "My Contacts" application is implemented in a separate method per table. The table definition for the company table is provided in Listing 5.9. In this method two Scout aspects are of interest.

The first Scout feature used is the absence of a `COMMIT` statement after the two `INSERT INTO` statements. This is possible, as all Scout service calls run in a transaction context that is transparent to the Scout developer. If a service method exits without errors, the enclosing transaction is committed. And if any runtime exception occurs in a service call the Scout runtime framework performs a rollback.

The second feature is the parameter binding used in the `INSERT INTO` statements. When SQL statements are executed using any of the static `SQL` methods, an internal statement processor replaces all bind variables found in the provided statement string. In Scout, SQL bind variables

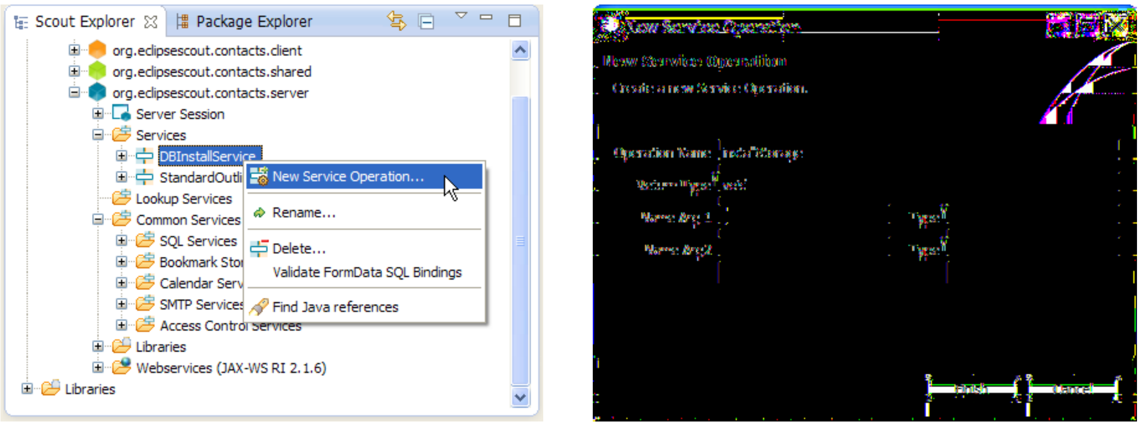


Figure 5.19: Add the service operation to create the DB schema.

Listing 5.8: The `installStorage` service operation to create the database schema for the "My Contacts" application. New tables are created if they are not found by `getExistingTables` in the existing schema.

```
public class DBInstallService extends AbstractService implements ↘
 →IDBInstallService {

  private boolean m_doSetup = true;

  public void setDoSetup(boolean doSetup) {
    m_doSetup = doSetup;
  }

  @Override
  public void installStorage() throws ProcessingException {
    boolean addInitialData = true;
    if (m_doSetup) {
      Set<String> tables = getExistingTables();
      createCompanyTable(tables, addInitialData);
      createPersonTable(tables, addInitialData);
      createUsersParamTable(tables, addInitialData);
    }
  }

  private Set<String> getExistingTables() throws ProcessingException {
    Object[][] existingTables = SQL.select("SELECT tablename FROM sys.↘
     →systables");
    HashSet<String> result = new HashSet<String>(existingTables.length);
    for (Object[] row : existingTables) {
      String table = (row[0] + "").toUpperCase();
      result.add(table);
    }
    return result;
  }
}
```

Listing 5.9: Setting up the `COMPANY` table of the ”My Contacts” application.

```
private void createCompanyTable(Set<String> tables, boolean addInitialData) ↘
 →throws ProcessingException {
 if (!tables.contains("COMPANY")) {
  SQL.insert("CREATE TABLE COMPANY ("
    + " company_id VARCHAR(64) NOT NULL CONSTRAINT COMPANY_PK PRIMARY KEY,↘
     → "
    + " name VARCHAR(64), "
    + " location VARCHAR(64), "
    + " url VARCHAR(64))");

  if (addInitialData) {
   SQL.insert("INSERT INTO COMPANY (company_id, name, location, url) "
     + "VALUES (:company_id, ’Alice’’s Adventures in Wonderland’, ’London↘
      →, England’, ’http://en.wikipedia.org/wiki/Alice%27↘
      →s_Adventures_in_Wonderland’)",
     new NVPair("company_id", UUID.randomUUID().toString()));

   SQL.insert("INSERT INTO COMPANY (company_id, name, location, url) "
     + "VALUES (:company_id, ’BSI Business Systems Integration AG’, ’↘
      →Daettwil, Baden, Switzerland’, ’http://www.bsiag.com’)",
     new NVPair("company_id", UUID.randomUUID().toString()));
  }
 }
}
```

need to use the pattern `:<variable name>`. The values for the bind variables can then be provided in the form of additional arguments. In the concrete example of Listing 5.9, the content for the bind variable `:company_id` is provided as a `NVPair` object. *NVPair*s are the simplest possible form to represent bind variables. The first constructor argument is the variable name of the bind variable, in this case `company_id`. The actual content of the bind variable is provided in the form of an object. Here, the Java runtime class `UUID.randomUUID().toString()` is used to create a new company key.

Setting up the person table and the user parameter table is defined according to Listing 5.10 and Listing 5.11. To create a data object for the persons day of birth, the Scout utility class `DateUtility` is used.

The only piece missing to setup the database is calling the `installStorage` operation during the startup of the ”My Contacts” server application. The proper way to implement such a scenario is to schedule an installation job in the `ServerApplication` class of the Scout server. In the ”My Contacts” application this is implemented according to Listing 5.12. To create a server job, a new server session object needs to be obtained first. However, during the startup time of the server we do not have any logged in users yet. That is why the session is created for the backend subject representing the server application. Using this `serverSession` object, the `installJob` can be created. In it’s `runTransaction` method we can then call the `installStorage` operation to setup the ”My Contacts” database.

Listing 5.10: Setting up the `PERSON` table of the ”My Contacts” application.

```
private void createPersonTable(Set<String> tables, boolean addInitialData) ↘
  →throws ProcessingException {
  if (!tables.contains("PERSON")) {
    SQL.insert("CREATE TABLE PERSON ("
        + " person_id VARCHAR(64) NOT NULL CONSTRAINT PERSON_PK PRIMARY KEY, "
        + " company_id VARCHAR(64), "
        + " first_name VARCHAR(64), "
        + " last_name VARCHAR(64), "
        + " headline VARCHAR(512), "
        + " location VARCHAR(512), "
        + " date_of_birth DATE, "
        + " picture_url VARCHAR(512), "
        + " CONSTRAINT COMPANY_FK FOREIGN KEY (company_id) REFERENCES COMPANY ↘
          →(company_id))");

    if (addInitialData) {
      SQL.insert("INSERT INTO PERSON (person_id, first_name, headline, ↘
        →location, company_id, picture_url, date_of_birth)"
          + "VALUES (:person_id, 'Alice', 'The curious girl', 'Daresbury, ↘
            →Cheshire, England', "
          + "(SELECT company_id FROM COMPANY WHERE name = 'Alice''s Adventures↘
            → in Wonderland'), 'http://www.uergsel.de/uploads/Alice.png', :dob↘
            →)",
          new NVPair("person_id", UUID.randomUUID().toString()),
          new NVPair("dob", DateUtility.parse("26.11.1865", "dd.mm.yyyy")));
    }
  }
}
```

Listing 5.11: Setting up the `USERS_PARAM` table of the ”My Contacts” application.

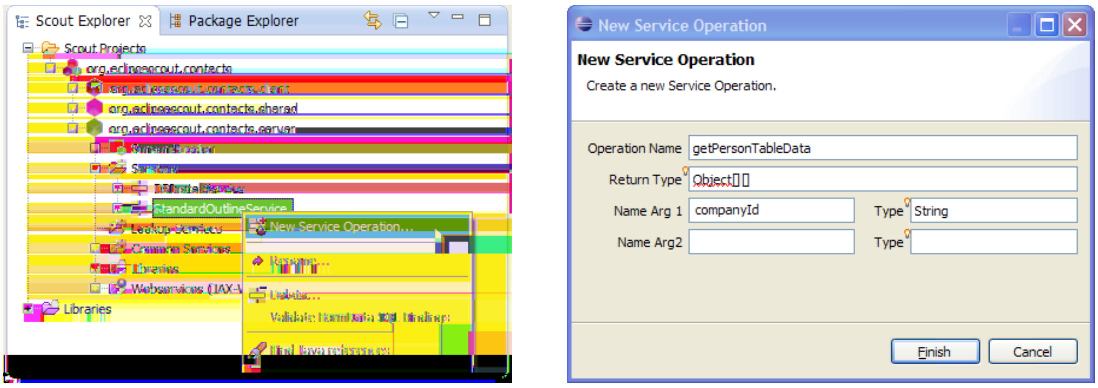
```
private void createUsersParamTable(Set<String> tables, boolean ↘
  →addInitialData) throws ProcessingException {
  if (!tables.contains("USERS_PARAM")) {
    SQL.insert(" CREATE TABLE USERS_PARAM ("
        + " username VARCHAR(32) NOT NULL, "
        + " param VARCHAR(32) NOT NULL, "
        + " value VARCHAR(512), "
        + " CONSTRAINT PARAM_PK PRIMARY KEY (username, param)"
        + ")");

    if (addInitialData) {
      // nop
    }
  }
}
```

Listing 5.12: Scheduling the database installation in the `start` method of the `ServerApplication` class during the server application startup.

```
public class ServerApplication implements IApplication {
  private static IScoutLogger logger = ScoutLogManager.getLogger(↘
    →ServerApplication.class);

  @Override
  public Object start(IApplicationContext context) throws Exception {
    ServerSession serverSession = SERVICES.getService(↘
      →IServerSessionRegistryService.class).newServerSession(ServerSession.↘
      →class, Activator.getDefault().getBackendSubject());

    ServerJob installJob = new ServerJob("Install the contacts DB schema", ↘
      →serverSession) {
      @Override
      protected IStatus runTransaction(IProgressMonitor monitor) {
        try {
          SERVICES.getService(IDBInstallService.class).installStorage();
          logger.info("Contacts DB schema successfully created");
        }
        catch (Throwable t) {
          return new Status(IStatus.ERROR, Activator.PLUGIN_ID, "Error while ↘
            →installing the contacts DB schema", t);
        }

        return Status.OK_STATUS;
      }
    };
    installJob.schedule();
    installJob.join(20000);

    logger.info("Contacts server application started");
    return EXIT_OK;
  }
}
```

Listing 5.13: The logging configuration in the Scout server's `config.ini` file.

```
eclipse.consoleLog=true
org.eclipse.scout.log=eclipse
org.eclipse.scout.log.level=INFO
```



New Service Operation

Create a new Service Operation.

Operation Name: getPersonTableData

Return Type: Object[][]

Name Arg 1: companyId Type: String

Name Arg2: Type:

Figure 5.20: Add the service operation to fetch the data for the person table.

5.5.2 Scout Logging

An additional Scout topic that is touched in this `ServerApplication` class is the Scout logging. As shown in Listing 5.12, a static `logger` object is created using Scout's `ScoutLogManager` class. Events can then be logged with the `logger.info` method where `info` refers to the log level attached to this message. Similar to the database setup described above, the logging setup can be defined in `config.ini` file. The default setup defined for the development product is provided in Listing 5.13. Further information regarding logging in Eclipse Scout is available in the Scout wiki[12].

5.6 Fetching Data from the Database

In the section before, we have implemented the access to the Derby database for the Scout server. And during the server startup, the application's database schema can now be created and populated with some initial data.

Here, this schema and the implemented database access is used to fetch the data for the client application's person page and the company page. The best place to implement these data provider methods is in the server's `StandardOutlineService` class. This class has been created by the Scout SDK during the initial project creation step. It is meant to hold the operations that fetch the data for populating the elements visible in the outline tree and outline pages.

For the "My Contacts" application, we first create the `getPersonTableData` operation as shown in Figure 5.20. In the wizard dialog enter "getPersonTableData" into the *Operation Name* field. For the return type we simply use a two dimensional object array. As the person page is also displayed under the company page, we need to have a way to only return the persons working for a specific company. To allow for this use case, we add the parameter "companyId" as the first argument to the `getPersonTableData` operation before we close the wizard with FINISH button.

The next operation we need is the `getCompanyTableData` method. You may use the same creation steps as described above for the person table page data. But for fetching company table data we do not need an additional argument. The company page in the "My Contacts" application

---

[12] Logging in Eclipse Scout: http://wiki.eclipse.org/Scout/Concepts/Logging.

Listing 5.14: Fetching the table data for the person and the company page of the "My Contacts" application. The list of persons is restricted to employees of a specific company if a company id parameter is provided.

```
public class StandardOutlineService extends AbstractService implements ↘
  →IStandardOutlineService {

  @Override
  public Object[][] getPersonTableData(String companyId) throws ↘
    →ProcessingException {
    String stmt = "SELECT person_id, first_name, last_name, headline FROM ↘
      →PERSON";

    if (StringUtility.isNullOrEmpty(companyId)) {
      return SQL.select(stmt);
    }

    return SQL.select(stmt + " WHERE company_id = :companyId",
        new NVPair("companyId", companyId));
  }

  @Override
  public Object[][] getCompanyTableData() throws ProcessingException {
    return SQL.select("SELECT company_id, name, location FROM COMPANY");
  }
}
```

will always show all available companies.

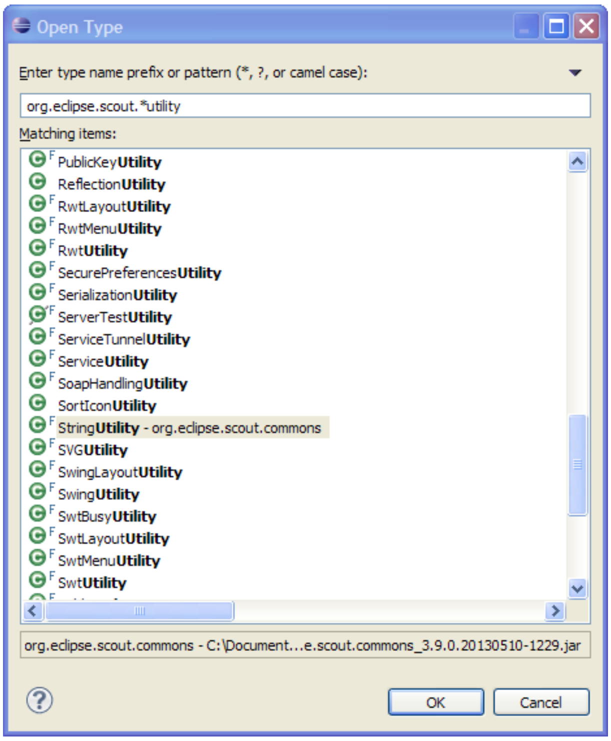
For the actual implementation of the two data fetching operations the code provided in Listing 5.14 is used. The implementation is straight forward and almost trivial. However, we can use the `getPersonTableData` example to introduce one of the many Scout utility classes.

The class `StringUtility` is one of the many utility classes provided by the Scout framework. Here, it is used for the typical null or empty check. To get a quick overview, hit the <Ctrl><Shift><T> key combination. In the type dialog that appears enter `org.eclipse.scout.*Utility` into the pattern field. This will display the complete list of the Scout utility classes as shown in Figure 5.21.

5.7 Creating the Person Form

In this section we will create the person form that is used to display and edit the persons stored in the "My Contacts" application. To add the person form use the Scout SDK *New Form* wizard as shown in Figure 5.22. In the first wizard step you just need to enter "Person" as a new translated text into the *Name* field and add "ui.forms" as the sub-package name in the corresponding wizard field. Then click the Finish button.

The wizard will create the necessary artefacts in the application's client, the shared and the server plugin projects. On the client side the `PersonForm` class with the necessary form handlers is created, in the shared part, the `PersonFormData` transfer object is added. On the server side,



Open Type

Enter type name prefix or pattern (*, ?, or camel case):

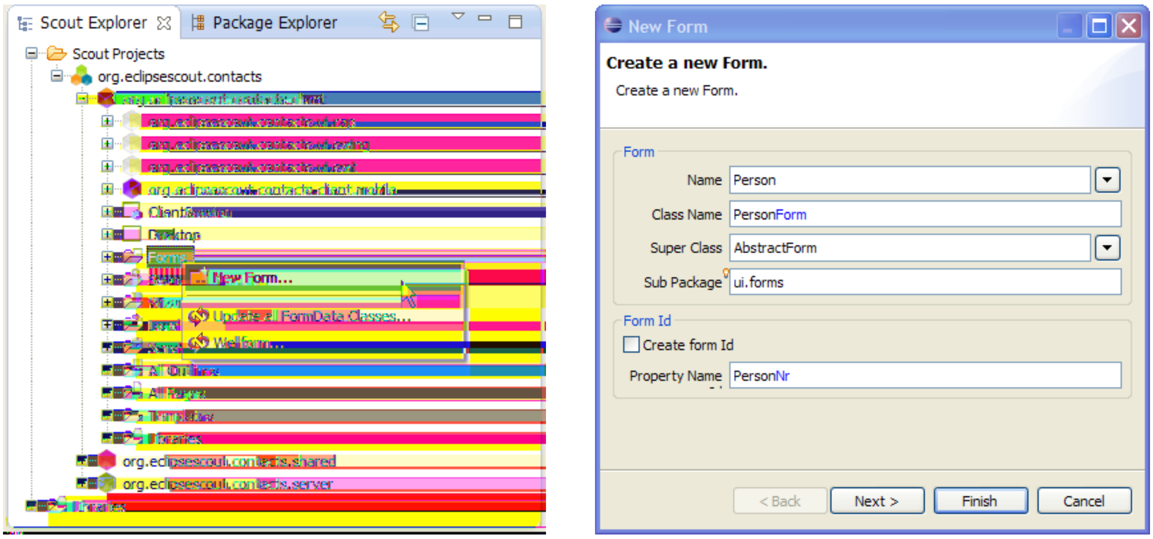
org.eclipse.scout.*utility

Matching items:

PublicKeyUtility
ReflectionUtility
RwtLayoutUtility
RwtMenuUtility
RwtUtility
SecurePreferencesUtility
SerializationUtility
ServerTestUtility
ServiceTunnelUtility
ServiceUtility
SoapHandlingUtility
SortIconUtility
StringUtility - org.eclipse.scout.commons
SVGUtility
SwingLayoutUtility
SwingUtility
SwtBusyUtility
SwtLayoutUtility
SwtMenuUtility
SwtUtility

org.eclipse.scout.commons - C:\Document...e.scout.commons_3.9.0.20130510-1229.jar

Figure 5.21: Accessing the Scout utility classes with the pattern `org.eclipse.scout.*Utility`.



Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.contacts
New Form...
Update all FormData Classes...
org.eclipsescout.contacts.shared
org.eclipsescout.contacts.server

New Form

Create a new Form.

Create a new Form.

Form

Name: Person

Class Name: PersonForm

Super Class: AbstractForm

Sub Package: ui.forms

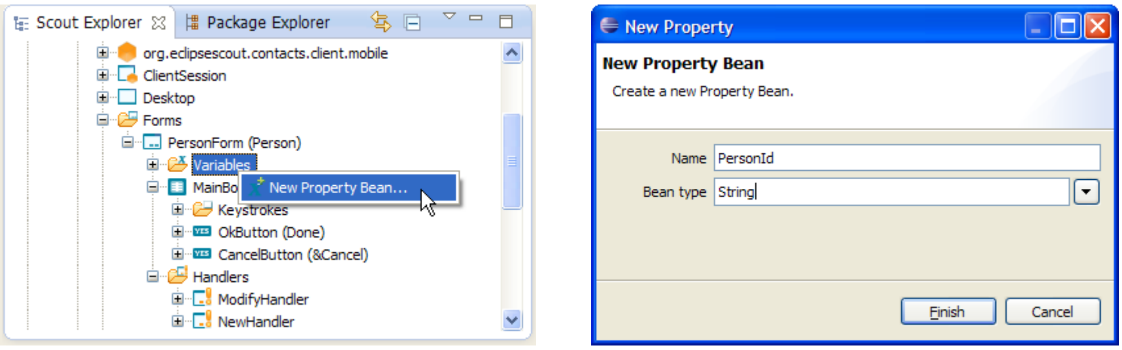
Form Id

☐ Create form Id

Property Name: PersonNr

< Back
Next >
Finish
Cancel

Figure 5.22: Add the person form.



New Property

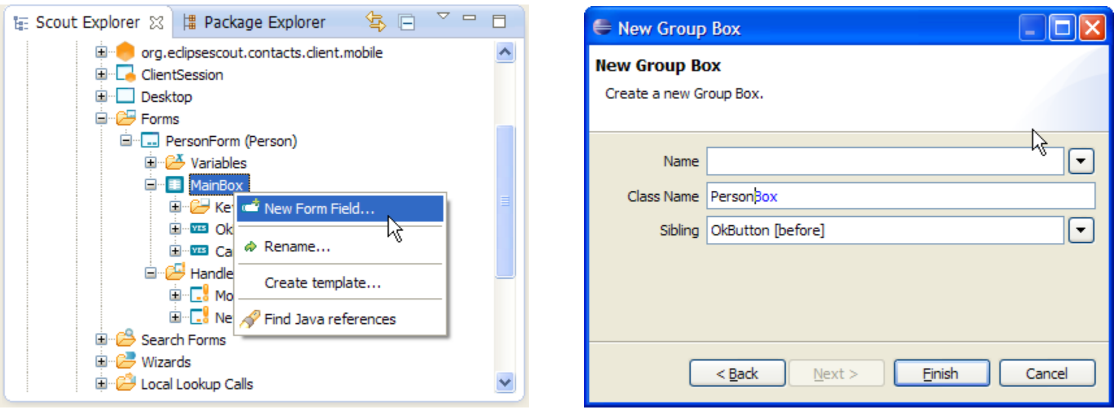
New Property Bean

Create a new Property Bean.

Name: PersonId

Bean type: String

Figure 5.23: Add the *PersonId* variable to the person form.



New Group Box

New Group Box

Create a new Group Box.

Name:

Class Name: PersonBox

Sibling: OkButton [before]

Figure 5.24: Add the first group box field to the person form.

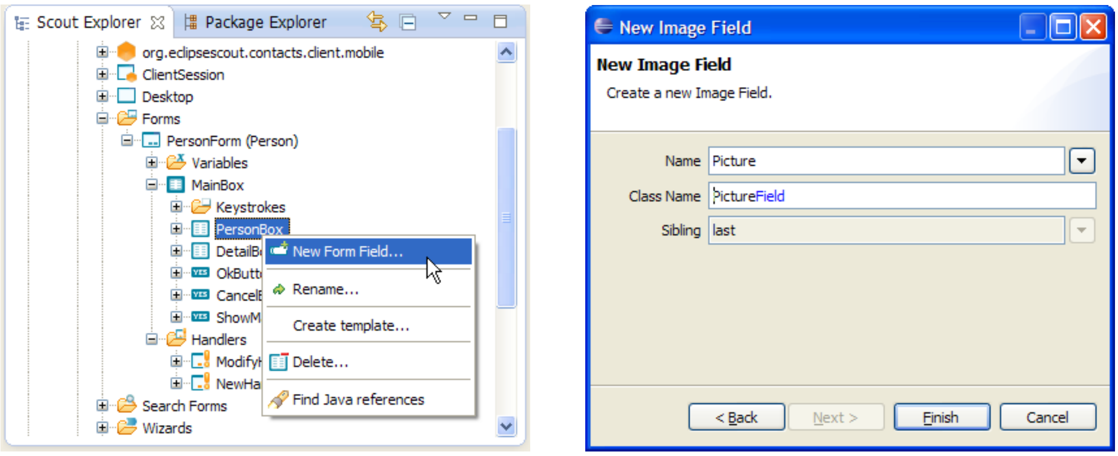
a `PersonProcessService` with all necessary service operations referenced by the form handlers is implemented.

To hold the persons primary key in the person form we also need to add a corresponding variable. For this, use the *New Property Bean* wizard as shown in Figure 5.23. In the wizard dialog, enter "PersonId" into the *Name* field and pick `String` from the dropdown list provided in the *Bean type* field. Then, click the FINISH button to close the wizard.

5.7.1 Creating the Form Layout with From Fields

As a next step we create the main layout structure of the person form. According to the screenshot of the person form shown in Figure 5.1 the form is organized into an upper box including the first name, the last name and a picture field. In the "Details" box in the lower half of the form some additional fields are found, such as the date of birth field. The bottom of the form holds a "Show Map ..." link. We start with the general layout by adding the `PersonBox` according to Figure 5.24. And again on the person's from *MainBox* node we add a second group box field using the label "Detail". The link is added with the *New Form Field* wizard too. In the first wizard step select the type `LinkButton` and use a new translated text "Show Map ..." for the *Name* field

We now add the form fields listed below using the the *New Form Field* wizard multiple times.



New Image Field

Create a new Image Field.

Name: Picture

Class Name: PictureField

Sibling: last

Figure 5.25: Add the picture field to the first group box of the person form.

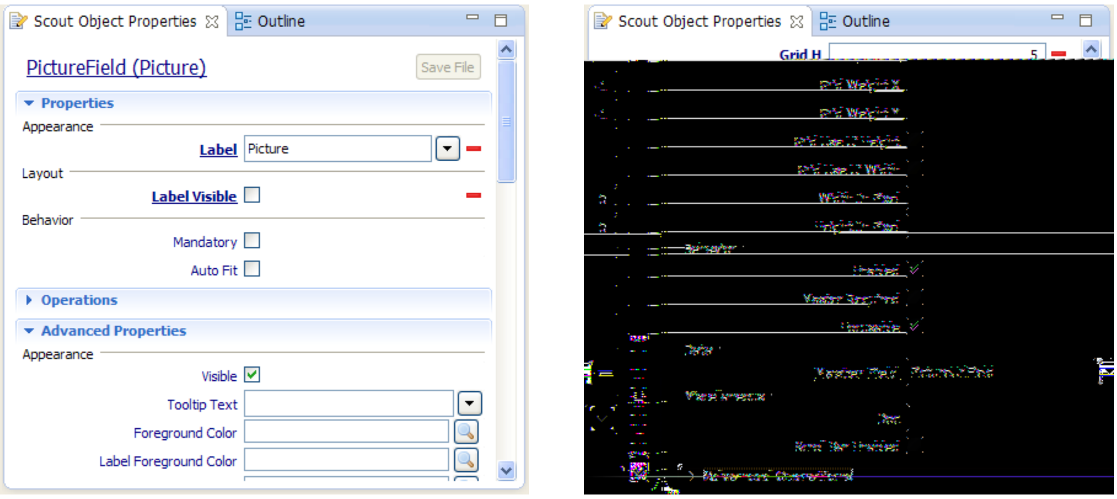
Most fields are of type `StringField` and different field types are separately indicated.

- PersonBox
  - ”First Name”
  - ”Last Name”
  - PictureUrlField
  - PictureField (ImageField)
- ”Detail” DetailBox
  - ”Headline”
  - ”Location”
  - ”Date of Birth” (DateField)

As an example for adding the form fields, the process is illustrated in Figure 5.25 for the creation of the picture field. In the field list we need to set non default properties for the *PictureUrlField* field and the *PictureField* field. The picture URL field will hold an URL pointing to the picture of the person to be displayed. As in the person form only the picture is to be displayed but not the picture URL, we need to make this field invisible. For this, select the *PictureUrlField* node in the Scout Explorer and then untick the *Visible* field in the Scout Object Properties. Note that to access the visibility property you need to open the section *Advanced Properties* first.

As the configuration of the picture field is more complex than the other fields, the changed properties are shown in the screenshots provided in Figure 5.26. First, no label is shown for the picture field as shown in the unticked *Label Visible* field of the Scout Object Properties. Then, property *Grid H* is set to 5. This results in the picture to cover the vertical space of 5 form fields.

Finally, we want the picture to be refreshed whenever the content of the picture URL is changed. For this, the property *Master Field* is set to the `PictureUrlField`. The implementation of the corresponding method `execChangedMasterValue` is provided in Listing 5.15. As we can see, the Scout helper class `IOUtility` is used to read the image content from the provided URL. This content is then used to assign the image content with the image field’s method



PictureField (Picture)

Properties

Appearance

Label: Picture

Layout

Label Visible ☐

Behavior

Mandatory ☐

Auto Fit ☐

Operations

Advanced Properties

Appearance

Visible ☑

Tooltip Text

Foreground Color

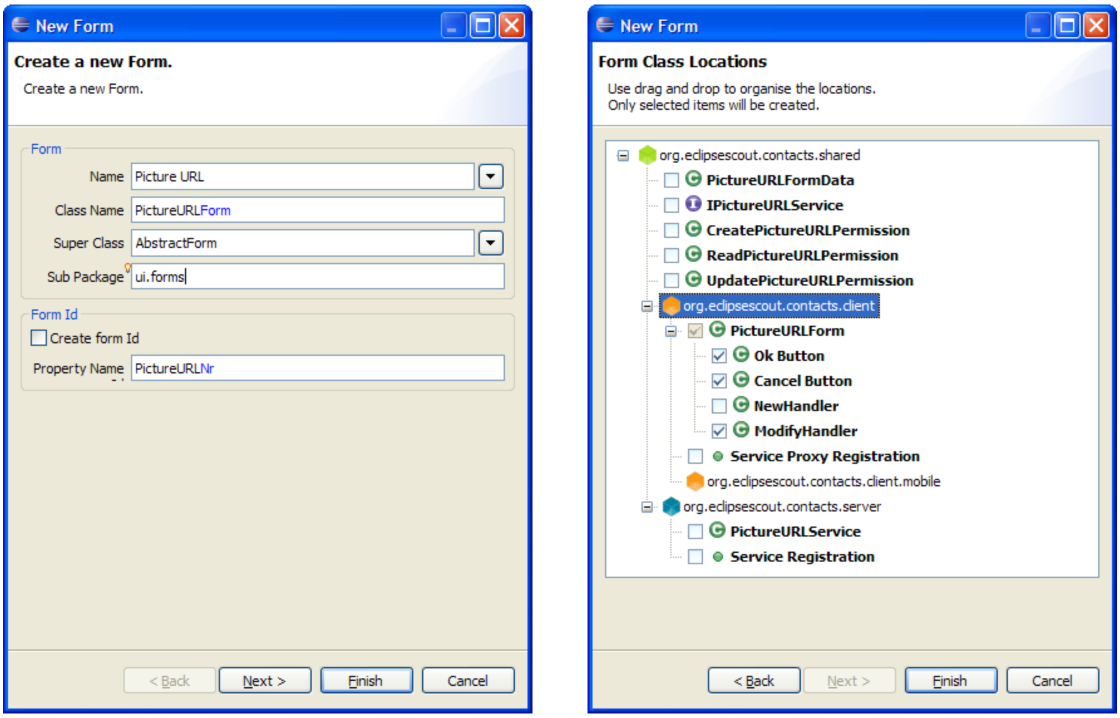
Label Foreground Color

Grid H: 5

Figure 5.26: Set the properties for the picture field of the person form.

Listing 5.15: The behaviour of the person form's `PictureField` is triggered by changes in field `PictureUrlField`.

```
@Order(40.0)
public class PictureField extends AbstractImageField {
  @Override
  protected Class<? extends IValueField> getConfiguredMasterField() {
    return PictureUrlField.class;
  }

  @Override
  protected void execChangedMasterValue(Object newMasterValue) throws ↘
    →ProcessingException {
    try {
      URL url = new URL((String) newMasterValue);
      setImage(IOUtility.getContent(url.openStream()));
      setAutoFit(true);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```



Create a new Form.

Create a new Form.

Form

Name: Picture URL

Class Name: PictureURLForm

Super Class: AbstractForm

Sub Package: ui.forms

Form Id

☐ Create form Id

Property Name: PictureURLNr

Form Class Locations

Use drag and drop to organise the locations.
Only selected items will be created.

- org.eclipsescout.contacts.shared
  - ☐ PictureURLFormData
  - ☐ IPictureURLService
  - ☐ CreatePictureURLPermission
  - ☐ ReadPictureURLPermission
  - ☐ UpdatePictureURLPermission
- org.eclipsescout.contacts.client
  - ☑ PictureURLForm
    - ☑ Ok Button
    - ☑ Cancel Button
    - ☐ NewHandler
    - ☑ ModifyHandler
  - ☐ Service Proxy Registration
  - org.eclipsescout.contacts.client.mobile
- org.eclipsescout.contacts.server
  - ☐ PictureURLService
  - ☐ Service Registration

Figure 5.27: Add the URL editor form.

`setImage`. Method `setAutoFit` is called to adapt the picture to the dimensions available to the image field.

5.7.2 A simple Form to edit the Picture URL

To edit a person's picture link, we create a simple URL editor form as shown in Figure 5.27. As we only need this form to update the URL information of a person's picture field, we do not need any connectivity to the backend of the "My Contacts" application. That is why almost all form and service artefacts are deselected in the second wizard step shown on the right side of Figure 5.27.

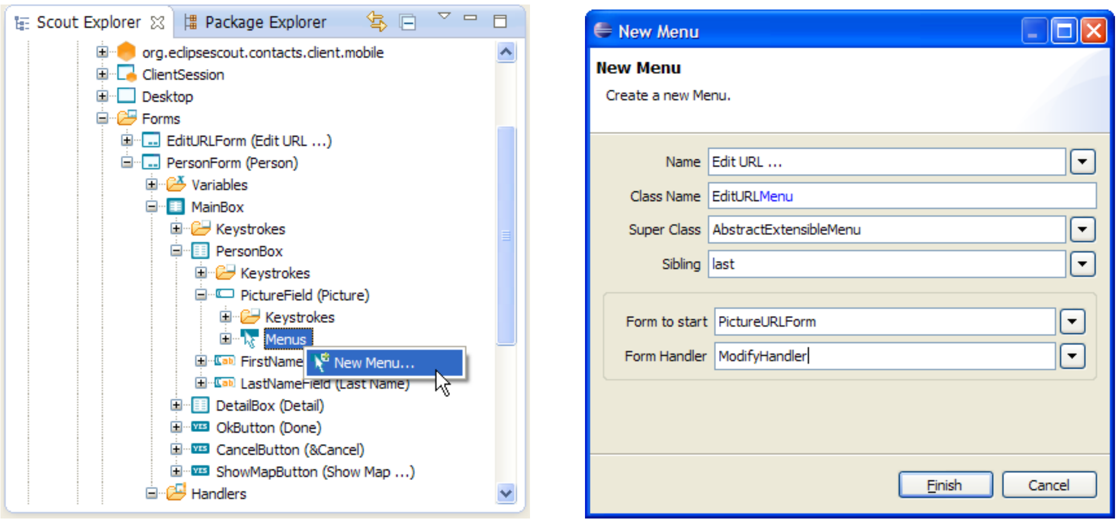
As this form only holds a single URL field, we omit the description of the creation of the URL editor form's content and provide the resulting Java code instead. In Listing 5.16 just the form's `MainBox` code is shown.

This form is then started via an "Edit URL ..." contextmenu on the image field. The creation of this contextmenu is shown in Figure 5.28. See Listing 5.17 for the actual implementation of the `execAction` for this contextmenu.

Once the edit URL form is started with `form.startModify()` the client waits in method `form.waitFor` until the form is closed by the user. If the user has changed any field content (the picture URL in our case) and closed the form with the OK button, the method `form.isFormStored` returns true, and we can copy the new URL from the editor form field into the picture URL field of the person form. Such a change will then trigger method `execChangedMasterValue` of the `PictureField` which in turn updates the image shown in the person form.

Listing 5.16: The UI structure of the `PictureURLForm` used to update the URL of the picture field in the person form. The rest of the class created by the Scout SDK has been omitted.

```java
public class PictureURLForm extends AbstractForm {
 @Order(10.0)
 public class MainBox extends AbstractGroupBox {

  @Order(10.0)
  public class UrlBox extends AbstractGroupBox {

   @Order(10.0)
   public class URLField extends AbstractStringField {

    @Override
    protected String getConfiguredLabel() {
     return TEXTS.get("URL");
    }
   }
  }
 }
}
```



New Menu

Create a new Menu.

Name: Edit URL ...

Class Name: EditURLMenu

Super Class: AbstractExtensibleMenu

Sibling: last

Form to start: PictureURLForm

Form Handler: ModifyHandler

Figure 5.28: Add the URL edit menu to the picture field.

Listing 5.17: The edit menu implemented in class `EditURLMenu` of the picture field. If the URL was changed the picture URL field of the person form is set accordingly in method `execAction`

```
@Order(10.0)
public class EditURLMenu extends AbstractExtensibleMenu {

  @Override
  protected String getConfiguredText() {
    return TEXTS.get("EditURL");
  }

  @Override
  protected void execAction() throws ProcessingException {
    PictureURLForm form = new PictureURLForm();
    form.startModify();
    form.waitFor();
    if (form.isFormStored()) {
      getPictureUrlField().setValue(form.getURLField().getValue());
    }
  }
}
```

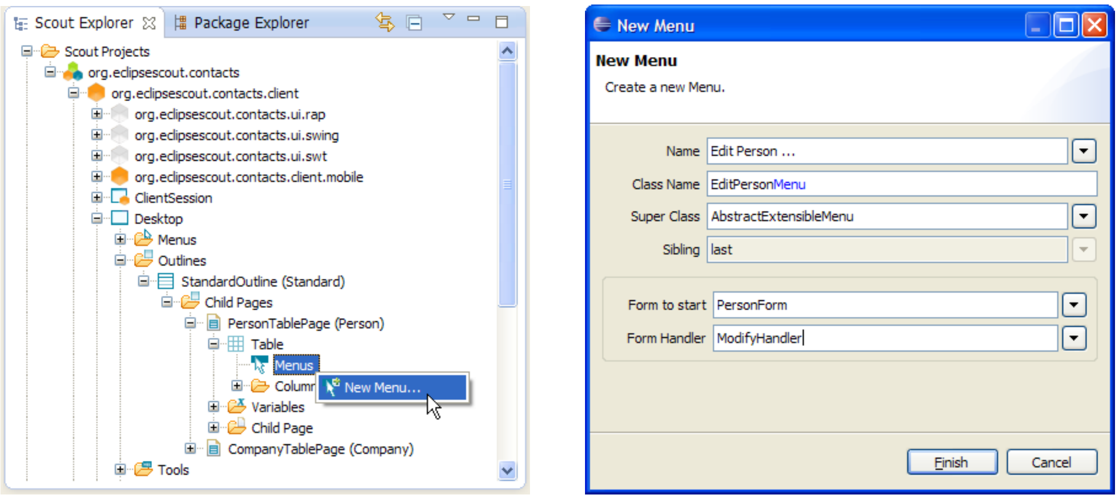
5.7.3 Linking the Person Form to the Person Page

The user of the "My Contacts" application wants to use the person form to show/edit attributes of a specific person. To support this use case, we need to link this form with the "My Contacts" application. As we already have person pages that show some of the person's attribute, we can now add context menus to this list to open/edit existing persons in the person form and to create new persons as well. This is achieved by using the *New Menu* wizard on the *Menus* node of the table of the person page according to Figure 5.29. In the *Name* field enter the new translated text "Edit Person ...". The form to start is the `PersonForm` and the `ModifyHandler` is the form handler to be used to start the form. As we have defined a meaningful primary key column on the person page and a matching variable is available for the person form, the Scout SDK wizard is generating the necessary code automatically. The implementation of the `execAction` method provided in Listing 5.18 works out of the box and should not need any manual tuning. Now, you may also add the "New Person ..." menu in the same way. Except that you pick the `NewHander` in the *New Menu* wizard instead of the modify handler.

To open the person form with a double click on a person row or by hitting the <ENTER> key, you may add a corresponding `execRowAction` on the person page. This method can be added to the person table by clicking on the green plus icon next to the operation *Exec Row Action* as shown in Figure 5.30. For the implementation of this method you may reuse the functionality implemented for the context menu according to Listing 5.19.

5.7.4 Adding the Company Smartfield

At the current stage of the "My Contacts" application, we have no option to manage the relationship between people and companies. To manage this relation, we now add a company smart field to the person form. This smart field will then hold the current assignment of the person represented in the person form.


Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.contacts
org.eclipsescout.contacts.client
org.eclipsescout.contacts.ui.rap
org.eclipsescout.contacts.ui.swing
org.eclipsescout.contacts.ui.swt
org.eclipsescout.contacts.client.mobile
ClientSession
Desktop
Menus
Outlines
StandardOutline (Standard)
Child Pages
PersonTablePage (Person)
Table
Menus
Columr
New Menu...
Variables
Child Page
CompanyTablePage (Company)
Tools
New Menu
Finish
Cancel

New Menu

Create a new Menu.

Name: Edit Person ...
Class Name: EditPersonMenu
Super Class: AbstractExtensibleMenu
Sibling: last
Form to start: PersonForm
Form Handler: ModifyHandler

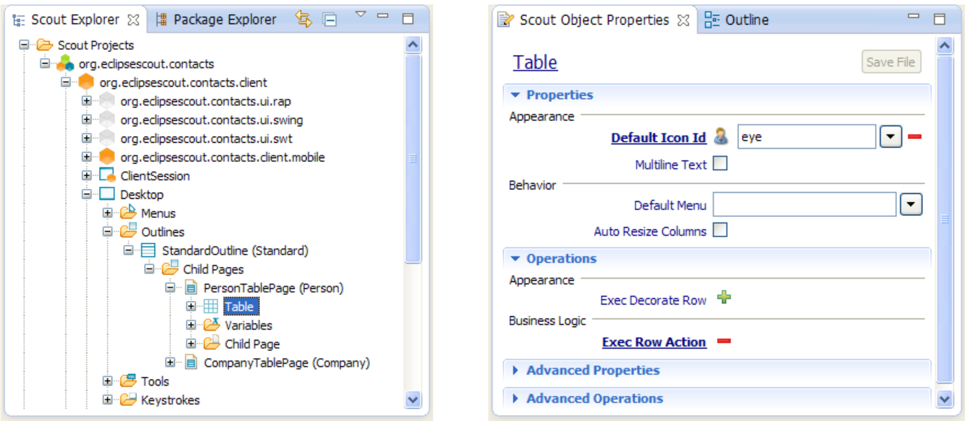
Figure 5.29: Add the Person edit menu to the person page.

Listing 5.18: The `EditPersonMenu` to edit the person selected in the person table. The person id taken from the corresponding (invisible) column is transferred to the person form before the form is started.

```
@Order(10.0)
public class EditPersonMenu extends AbstractExtensibleMenu {

  @Override
  protected String getConfiguredText() {
    return TEXTS.get("EditPerson");
  }

  @Override
  protected void execAction() throws ProcessingException {
    PersonForm form = new PersonForm();
    form.setPersonId(getPersonIdColumn().getSelectedValue());
    form.startModify();
    form.waitFor();
    if (form.isFormStored()) {
      reloadPage();
    }
  }
}
```

Listing 5.19: The `execRowAction` method on table pages is used to trigger an action when a row is selected with a double click or <ENTER>.

```
@Override
protected void execRowAction(ITableRow row) throws ProcessingException {
  getMenu(EditPersonMenu.class).execAction();
}
```



Default Icon Id: eye

Multiline Text ☐

Default Menu:

Auto Resize Columns ☐

Exec Decorate Row

Exec Row Action

Figure 5.30: Set the behaviour for the row level action on the person table.

Listing 5.20: The company lookup call with its `getConfiguredService` method in the application's shared plugin.

```
public class CompanyLookupCall extends LookupCall{

  private static final long serialVersionUID = 1L;

  @Override
  protected Class<? extends ILookupService> getConfiguredService() {
    return ICompanyLookupService.class;
  }
}
```

A Scout smart field can be viewed as user friendly dropdown field on steroids that also implements *search-as-you-type* to pick a specific entry. In the simplest case the smart field provides access to a small and locally provided list of key value pairs. But for the intended use in the "My Contacts" application, we will need to access a list of elements provided by the server that will be compiled dynamically at runtime.

To create the access to this list, we start with the creation of the company lookup call. As shown in Figure 5.31 the lookup call is added on the *Lookup Calls* folder under the green shared node of the "My Contacts" application. This opens the *New Lookup Call* wizard as shown in Figure 5.32. In the first wizard step, enter "Company" into the *Class Name* field and verify that the wizard step looks the same as the screenshot shown on the left hand side of Figure 5.32. Before the wizard is closed, click on the NEXT button to move to the second wizard step. As shown on the right hand side of Figure 5.32 the wizard will also create a corresponding `CompanyLookupService` on the application's server. We can now close this wizard with the FINISH button.

The `CompanyLookupCall` just created by the Scout SDK wizard is provided in Listing 5.20. As we can see, the only method implemented is `getConfiguredService` that points to the comind with the

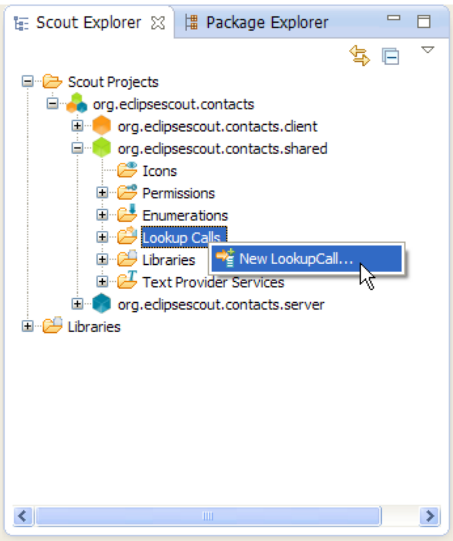
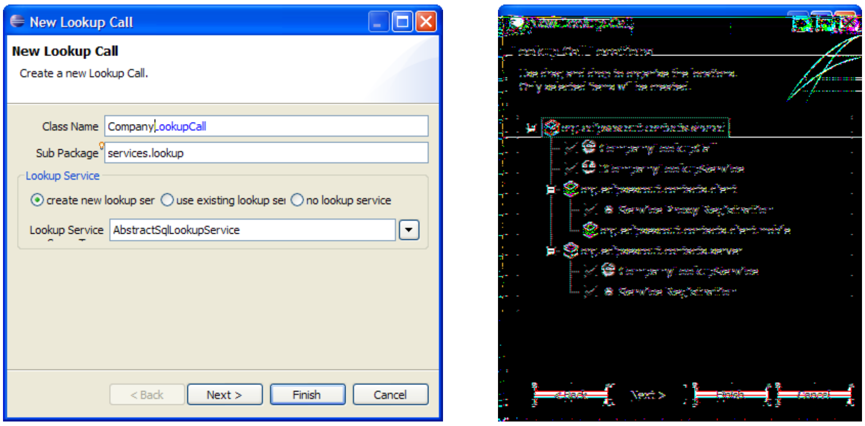


Figure 5.31: Add a lookup call to the applications shared node.



Class Name: CompanyLookupCall
Sub Package: services.lookup
Lookup Service: ● create new lookup ser ○ use existing lookup se ○ no lookup service
Lookup Service: AbstractSqlLookupService

Figure 5.32: The two wizard steps to enter the details of the company lookup call.

Listing 5.21: The company lookup service in the application's server plugin. The `key` and the `text` criteria are used to search for values by key or by the provided name substring. index

```
public class CompanyLookupService extends AbstractSqlLookupService implements↘
 → ICompanyLookupService {

  @Override
  protected String getConfiguredSqlSelect() {
    return "SELECT "
        + "company_id, "
        + "name "
        + "FROM COMPANY "
        + "WHERE 1=1 "
        + "<key> AND company_id = :key </key> "
        + "<text> AND UPPER(name) LIKE UPPER(:text||'%') </text> "
        + "<all> </all> ";
  }
}
```

included in the `SELECT` statement using the `<key>`, `<text>` and `<all>` tags as shown in the listing. The Scout runtime uses the `<key>`-block in cases where a specific key is already assigned to the smart field. The `<text>`-block is used as a query criteria to create the dynamic *search-as-you-type* hit list based on the (sub)string entered by the user so far. Finally, the `<text>`-block is used to define the hit list to be shown when the user does not enter any text into the smart field but clicks on the field's search icon instead. The bind variable `:key` and `:text` are provided by Scout and hold the value of the assigned key or the text entered into the smart field.

We are now ready to add the company smart field to the person form. To start the *New Form Field* wizard we use the context menu on the `DetailBox` of the person form as shown in Figure 5.34. In the first wizard step, we chose the *SmartField* entry as the field type and click the NEXT button. Then, we enter "Company" into the *Name* field as shown on the right hand side of Figure 5.34. Make sure that you select the *String* entry in the *Generic Type* field as we are using string values to identify companies in the "My Contacts" application. And in the *LookupCall* field, we can now select the *CompanyLookupCall* that we have just created before. Finally, the position of the new company smart field can be set in the *Sibling* field before the location field before the wizard is closed with the FINISH button.

The implementation of the company smart field created by the Scout SDK wizard is provided in Listing 5.22. A look at the implementation of the `CompanyField` class shows its simplicity and the wiring with the company lookup service.

5.7.5 Adding the Map Form

We now want to add the "Map" form shown in the front of Figure 5.1. The purpose of this form is to show a map corresponding to the address entered into the *location* field of the person form using the Google Maps Image API[13]. This also implies that only addresses that can correctly be parsed by the Maps Image API will lead to a useful map image.

---

[13] The Google Maps Image API can be used to fetch static map images: https://developers.google.com/maps/documentation/imageapis/.

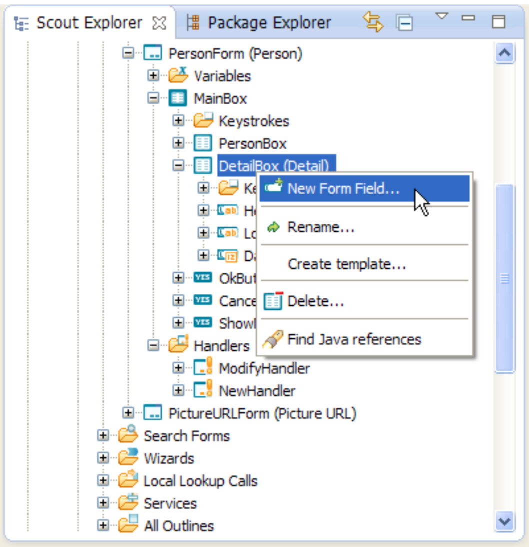
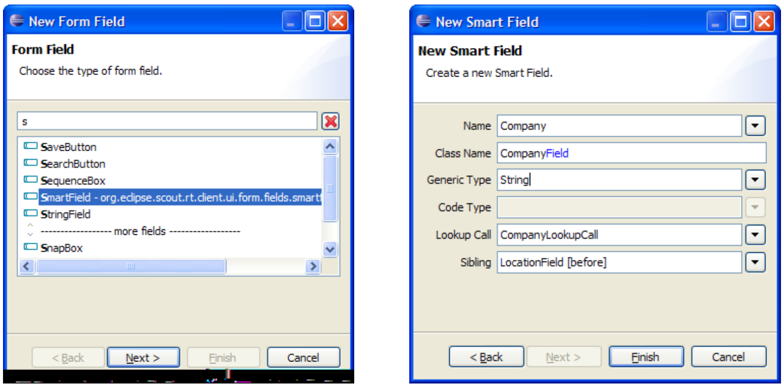


Figure 5.33: Add a smart field to the person form.


New Form Field
Form Field
Choose the type of form field.
s
SaveButton
SearchButton
SequenceBox
SmartField - org.eclipse.scout.rt.client.ui.form.fields.smart
StringField
------------------ more fields ------------------
SnapBox
< Back
Next >
Finish
Cancel
New Smart Field
New Smart Field
Create a new Smart Field.
< Back
Next >
Finish
Cancel

Name: Company
Class Name: CompanyField
Generic Type: String
Code Type:
Lookup Call: CompanyLookupCall
Sibling: LocationField [before]

Figure 5.34: Create the company smart field for the person form.

Listing 5.22: The smart field `CompanyField` of the person form and its wiring with the company lookup call.

```
@Order(20.0)
public class CompanyField extends AbstractSmartField<String> {

  @Override
  protected String getConfiguredLabel() {
    return TEXTS.get("Company");
  }

  @Override
  protected Class<? extends LookupCall> getConfiguredLookupCall() {
    return CompanyLookupCall.class;

  }
}
```

To create the maps form we start the *New Form* wizard and enter the new translated text ”Map” into the *Name* field of the first wizard step. Then, we click the NEXT button to configure the artefacts to be created by the wizard. For the map form we can use the configuration as shown on the right hand side of Figure 5.27 with the difference that we do not need the Ok button. Having deselected most artefacts, the wizard can be closed with the FINISH button.

After the form creation wizard has been closed, update the label of the cancel button to ”Close” in the button’s Scout Object Properties. Then, we can add an ”Address” variable to the form by starting the *New Property Bean* wizard on the *Variables* node of the newly created map form. In the property bean wizard, enter ”Name” into the *Name* field and set the *Bean type* field to *String*.

As the next step, the map image field is added to the from. For this, start the *New Form Field* wizard directly on the form’s *MainBox* node. In the first form field wizard step, select *ImageField* as the field type and click on the NEXT button. Before you can close the second wizard step with the FINISH button, enter ”Map” into the *Class Name* field. To set the properties of the new map field, select the *MapField* node below the main box node of the map form. In the `MapField`’s Scout Object Properties untick the *Label Visible* property and add an `execInitField` method by clicking on the green plus icon next to this operation. The configuration of the map field can then be completed in section *Advanced Properties*. Here, we set the *Grid H* property to 6 and update the *Width in Pixel* property and the *Height in Pixel* property to a value of 400 each.

To add the Java code to display the map in the image field, click on the *execInitField* link in the Scout Object Properties of the map field. According to the implementation provided in Listing 5.23, an URL for the Maps Image API is first constructed. This URL also contains the content of the map form’s address variable and the configured dimension of the map field. The map picture returned by the Google API is then read using `IOUtility.getContent` and directly fed into the image fields `setImage` method.

The last step involving the implementation of the map form feature is its integration into the person form. As visible on the lower left part of the person form shown in Figure 5.1, a *Show Map ...* link is available. We now need to add such a link to the person form and add the necessary wiring for opening the newly created map form if the user clicks on this link. As a first step, the *New Form Field* wizard is started on the *MainBox* node. In the first wizard step, select the

Listing 5.23: In the `execInitField` method of the map form the image content is fetched from the Google Maps API.

```
@Override
protected void execInitField() throws ProcessingException {
  String address = StringUtility.nvl(getAddress(), "");
  String size = "" + getConfiguredHeightInPixel() + "x" + ↘
    →getConfiguredWidthInPixel();
  String url = null;

  try {
    url = "http://maps.googleapis.com/maps/api/staticmap?center=" +
        URLEncoder.encode(address, "ISO-8859-1") +
        "&zoom=13&size=" + size + "&maptype=roadmap&sensor=false";
    setImage(IOUtility.getContent((new URL(url)).openStream()));
  }
  catch (Exception e) {
    setErrorStatus(new ProcessingStatus("Bad Link: '" + url + "', please ↘
      →check",
        ProcessingStatus.ERROR));
    setImage(null);
    e.printStackTrace();
  }
}
```

LinkButton from the available field types and click the NEXT button to load the second wizard step. Here, just enter the new translated text ”Show Map ...” into the *Name* field and close the wizard with the FINISH button.

To add the necessary wiring code to the link button double click the *ShowMapButton* node in the Scout Explorer and implement its `execClickAction` method. As shown in Listing 5.24, we only need to create a new map form in the click action, set its address variable and open the form with `form.startModify`.

5.8 Managing Person Data on the Server Side

In the ”My Contacts” client application the person form is started via two separate form handlers called by the person page context menus. The new handler is used to start the form in the mode to create new persons while the modify handler is implemented to start the form for updating existing persons. When we look at the implementation of the modify handler provided in Listing 5.25, we see the two methods `execLoad` and `execStore`.

As already described in Section 3.2.3 of the ”Hello World” tutorial, `execLoad` is called by the Scout framework the the form is started. Similarly, method `execStore` is called by the Scout framework after the user has clicked the OK button of a form. In both methods, a corresponding service operation of the `PersonService` is called. From the form handler’s `execLoad` method the method `load` of the person service is called. And the form handler’s `execStore` method calls the person service’s method `store` as shown in Listing 5.26.

For the `load` method in Listing 5.26 it is assumed that the in the provided parameter

Listing 5.24: The edit menu implemented in class `EditURLMenu` of the picture field. If the URL was changed the picture URL field of the person form is set accordingly in method `execAction`

```
  @Order(50.0)
  public class ShowMapButton extends AbstractLinkButton {

    @Override
    protected String getConfiguredLabel() {
      return TEXTS.get("ShowMap");
    }

    @Override
    protected void execClickAction() throws ProcessingException {
      MapForm mapForm = new MapForm();
      mapForm.setAddress(getLocationField().getValue());
      mapForm.startModify();
    }
  }
}
```

Listing 5.25: The `execLoad` and the `execStore` methods of the person form's modify handler.

```
public class ModifyHandler extends AbstractFormHandler {

  @Override
  public void execLoad() throws ProcessingException {
    IPersonService service = SERVICES.getService(IPersonService.class);
    PersonFormData formData = new PersonFormData();
    exportFormData(formData);
    formData = service.load(formData);
    importFormData(formData);
    setEnabledPermission(new UpdatePersonPermission());
  }

  @Override
  public void execStore() throws ProcessingException {
    IPersonService service = SERVICES.getService(IPersonService.class);
    PersonFormData formData = new PersonFormData();
    exportFormData(formData);
    formData = service.store(formData);
  }
}
```

Listing 5.26: The `load` and the `store` methods of the server's `PersonService`.

```
public class PersonService extends AbstractService implements IPersonService ↘
  →{
  @Override
  public PersonFormData load(PersonFormData formData) throws ↘
    →ProcessingException {
    if (!ACCESS.check(new ReadPersonPermission())) {
      throw new VetoException(TEXTS.get("AuthorizationFailed"));
    }

    SQL.selectInto("SELECT "
        + "picture_url, "
        + "first_name, "
        + "last_name, "
        + "company_id, "
        + "headline, "
        + "location, "
        + "date_of_birth "
        + "FROM PERSON "
        + "WHERE PERSON_ID = :personId "
        + "INTO "
        + ":pictureUrl, "
        + ":firstName, "
        + ":lastName, "
        + ":company, "
        + ":headline, "
        + ":location, "
        + ":dateOfBirth",
        formData);

    return formData;
  }

  @Override
  public PersonFormData store(PersonFormData formData) throws ↘
    →ProcessingException {
    if (!ACCESS.check(new UpdatePersonPermission())) {
      throw new VetoException(TEXTS.get("AuthorizationFailed"));
    }
    SQL.update("UPDATE PERSON SET "
        + "picture_url = :pictureUrl, "
        + "first_name = :firstName, "
        + "last_name = :lastName, "
        + "headline = :headline, "
        + "company_id = :company, "
        + "location = :location, "
        + "date_of_birth = :dateOfBirth "
        + "WHERE person_id = :personId ",
        formData);

    return formData;
  }
}
```

Listing 5.27: The `create` method of the server's `PersonService`.

```
public class PersonService extends AbstractService implements IPersonService ↘
  →{
 @Override
 public PersonFormData create(PersonFormData formData) throws ↘
   →ProcessingException {
   if (!ACCESS.check(new CreatePersonPermission())) {
     throw new VetoException(TEXTS.get("AuthorizationFailed"));
   }

   if (StringUtility.isNullOrEmpty(formData.getPersonId())) {
     formData.setPersonId(UUID.randomUUID().toString());
   }

   SQL.insert("INSERT INTO PERSON (person_id) "
       + "SELECT :personId "
       + "FROM PERSON "
       + "WHERE person_id = :personId "
       + "HAVING count(*) = 0",
       formData);

   return store(formData);
 }
}
```

`formData` the person's id is available as a search criteria. It might be interesting to note that for the SQL `SELECT` and `UPDATE` statements we can provide the complete form data to use Scout's support for bind variables.

The implementation of the person form's new handler is implemented similar to it's modify handler. Instead of calling the service operations `load` and `store`, the methods `prepareCreate` and `create` are used. In principle, method `prepareCreate` is not needed for the "My Contacts" application. As its implementation has been created by the Scout SDK wizard, it does not any harm when the method is left as it is.

In Listing 5.27 the implementation of the `PersonService`'s `create` method is provided. First, the implementation checks if the provided form data contains a person id. If the person has been entered manually a value for this id is missing and a new id is provided using `formData.setPersonId(UUID.randomUUID().toString())`. The only responsibility of the `INSERT` statement following the person id check is to make sure we will have a row in the `PERSON` table for the person to be created. To save all other form data parameters, we just need to call the previously implemented `store` method.

5.9 Creating the Company Form

Creating the company form and the necessary backend services is not described here. Instead this task is left as an exercise to the reader and in the text below some minimal guidelines are provided.

The main trick is to start with the *New Form* wizard to create the company form. This will

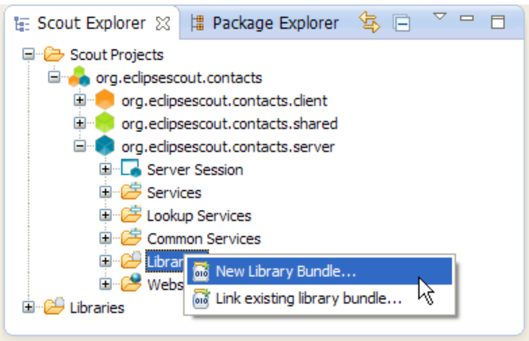

Scout Explorer
Package Explorer
Scout Projects
org.eclipsescout.contacts
org.eclipsescout.contacts.client
org.eclipsescout.contacts.shared
org.eclipsescout.contacts.server
Server Session
Services
Lookup Services
Common Services
Librar
Webs
Libraries
New Library Bundle...
Link existing library bundle...

Figure 5.35: Add a new bundle to hold the Scribe JARs.

create all necessary artefacts including the forms, necessary server services, and communication between the client and the server. Then, make sure you don't forget to add a variable `companyId` to the company form. To decide on the fields that need to be on the company form you may check the setup of the database schema provided in Listing 5.9. If in doubt about what to do, please refer to the procedure used to create the person form.

In case you get lost completely, you may download the "My Contacts" application from this books Github repository as described in the Scout wiki[14].

5.10 Adding the Scribe Library to the Application

To access data from sites such as LinkedIn, Xing, Google+ or Facebook, most social networks provide an API that requires user authentication. The current defacto standard for such authentication is the Open Authentication Protocol (OAuth)[15]

The big advantage of the OAuth standard is the access token based authentication. If an application such as the "My Contacts" example is in possession of an access token, it can fetch data from the hosting site or even act on the users behalf. It is important to note that the access token is just constructed of two values, the token and the secret. And both values are completely separate from the user's username and password credentials. This means that an access token may be safely stored in other applications. Should such an application's data get compromised, the username and password are still save. The user then has the option to remove the compromised access token.

For the "My Contacts" application we use the Scribe Java library[16] This library makes accessing social services very simple. At the same time the Scribe library is used here to demonstrate the integration of external Java libraries in Scout applications.

As we will use the Scribe library only on the Scout server side start the Scout SDK *New Library Bundle* wizard below the blue server node as shown in Figure 5.35. In the first wizard step shown on the left side of Figure 5.36 add the files `scribe-1.3.0.jar` and `commons-codec-1.8.jar`[17]

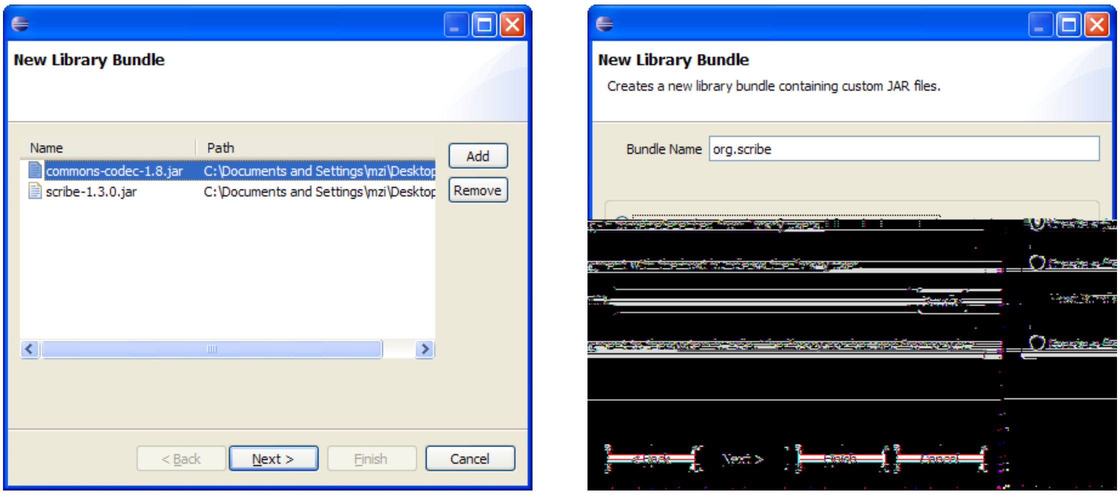
---

[14] Download and installation of the "My Contacts" application: `http://wiki.eclipse.org/Scout/Book/3.9#Download_and_Run_the_Scout_Sample_Applications`.

[15] For more information regarding the OAuth standard, see: `http://oauth.net/`.

[16] The Scribe OAuth Java library: `https://github.com/fernandezpablo85/scribe-java`.

[17] See the getting started page for more information regarding the two necessary JAR file: `https://github.com/fernandezpablo85/scribe-java/wiki/getting-started`.


New Library Bundle

Name	Path
commons-codec-1.8.jar	C:\Documents and Settings\mzi\Desktop
scribe-1.3.0.jar	C:\Documents and Settings\mzi\Desktop

New Library Bundle

Creates a new library bundle containing custom JAR files.

Bundle Name: org.scribe

Figure 5.36: Specify the JARs to be contained in the library bundle and the library name.

using the ADD button. Once the two JAR files are listed in the library list of the wizard click the NEXT button. On the second wizard step, enter ”org.scribe” into the *Bundle Name* field as shown on the right side of Figure 5.36. Then, close the wizard with the FINISH button.

The Scout SDK wizard then creates the corresponding library plugin and updates the server product files and the plugin dependencies of the applications server plugin accordingly. Once the wizard has completed the classes defined in the two JAR files can directly be accessed from the ”My Contact” application’s server plugin.

5.11 Integrating LinkedIn Access with Scribe

To access the LinkedIn data with Scribe a separate service `LinkedInService` is created in the *Services* folder under the blue server node. Before we add any service operations to the LinkedIn service, the service initialization method is implemented according to Listing 5.28. This initialization is not discussed further here. Instead, the interested reader is referred to the corresponding `LinkedInExample` on the Scribe web pages[18].

The first service operation `getAuthUrl` is added according to Figure 5.37. This operation will return the necessary information to create a request token and provides a link to open in a web browser to start the authentication against the LinkedIn account. For its implementation see Listing 5.29.

The next operations we add are used to refresh the users access token stored in the ”My Contacts” application and to refresh or download the users contacts stored in the user’s LinkedIn account. See Figure 5.38 for the necessary details for the creation of the two methods.

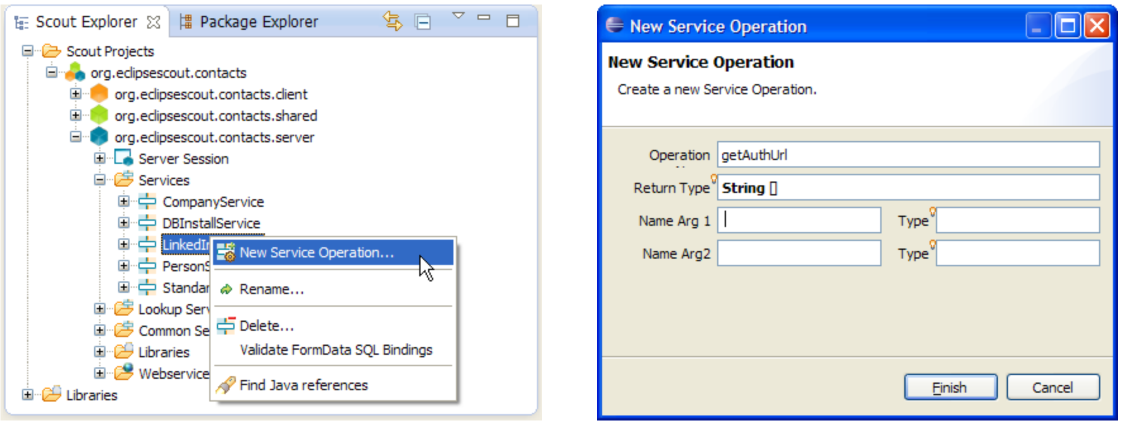
Method `refreshToken` is implemented according to Listing 5.30. Using the provided token parameters and the security code, the access token is created using `m_service.getAccessToken(...)`. Then, the user id of the currently logged in user is ob-

---

[18] For the Scribe `LinkedInExample.java` example class see: `https://github.com/fernandezpablo85/scribe-java/blob/master/src/test/java/org/scribe/examples/LinkedInExample.java`.

Listing 5.28: The `LinkedInService` service with it's `initializeService` method defined in the `IService` interface.

```
public class LinkedInService extends AbstractService implements ↘
  →ILinkedInService {
  private final static String LINKEDIN_TOKEN = "linkedin_access_token";
  private final static String LINKEDIN_SECRET = "linkedin_access_secret";
  private final static String LINKEDIN_CONNECTIONS = "http://api.linkedin.com/↘
    →v1/people/~/connections";

  private static IScoutLogger LOG = ScoutLogManager.getLogger(LinkedInService.↘
    →class);
  private OAuthService m_service = null;

  @Override
  public void initializeService(ServiceRegistration registration) {
    super.initializeService(registration);

    m_service = new ServiceBuilder()
        .provider(LinkedInApi.class)
        .apiKey("CiEgwWDkA5BFpNrc0RfGyVuSlOh4tig5kOTZ9q97qcXNrFl7zqk-↘
          →Ts7DqRGaKDCV")
        .apiSecret("dhho4dfoCmiQXrkw4yslork5XWLFnPSuMR-8↘
          →gscPVjY4jqFFHPYWJKgpFl4uLTM6")
        .build();
  }
```



Operation: getAuthUrl
Return Type: String[]

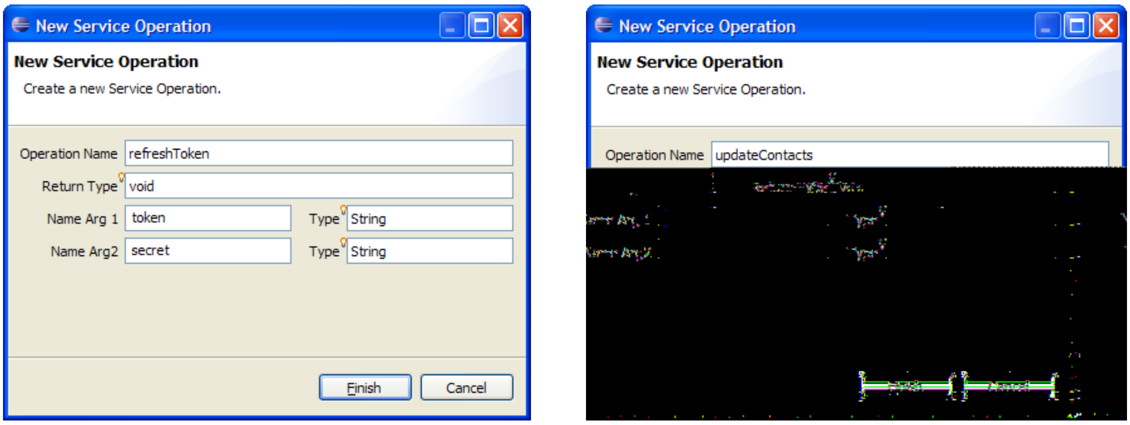
Figure 5.37: Add the operation to retrieve an authentication URL.

Listing 5.29: The `getAuthUrl` method of the LinkedIn service.

```
@Override
public String[] getAuthUrl() throws ProcessingException {
  Token requestToken = m_service.getRequestToken();
  String authLink = m_service.getAuthorizationUrl(requestToken);
  return new String[]{requestToken.getToken(), requestToken.getSecret(), ↘
    →authLink};
}
```

Listing 5.30: The `refreshToken` method is used to create a new access token to fetch data from the LinkedIn API.

```
@Override
public void refreshToken(String token, String secret, String securityCode) ↘
  →throws ProcessingException {
  // retrieve linked in access token
  Token accessToken = m_service.getAccessToken(new Token(token, secret), new↘
    → Verifier(securityCode));

  // make sure current user has param records for linkedin token
  String userName = ServerSession.get().getUserId();

  SQL.insert("INSERT INTO USERS_PARAM (username, param) ("
     + "SELECT :username as username, :param as param "
     + "FROM USERS_PARAM "
     + "WHERE username = :username AND param = :param HAVING count(*) = 0"
     + ")", new NVPair("username", userName), new NVPair("param", ↘
       →LINKEDIN_TOKEN));

  SQL.insert("INSERT INTO USERS_PARAM (username, param) ("
     + "SELECT :username as username, :param as param "
     + "FROM USERS_PARAM "
     + "WHERE username = :username AND param = :param HAVING count(*) = 0"
     + ")", new NVPair("username", userName), new NVPair("param", ↘
       →LINKEDIN_SECRET));

  // update param records with new access token
  SQL.update("UPDATE USERS_PARAM set value = :value WHERE param = :param AND↘
    → username = :username",
     new NVPair("value", accessToken.getToken()), new NVPair("username", ↘
       →userName), new NVPair("param", LINKEDIN_TOKEN));

  SQL.update("UPDATE USERS_PARAM set value = :value WHERE param = :param AND↘
    → username = :username",
     new NVPair("value", accessToken.getSecret()), new NVPair("username", ↘
       →userName), new NVPair("param", LINKEDIN_SECRET));
}
```



Operation Name: refreshToken
Return Type: void
Name Arg 1: token Type: String
Name Arg2: secret Type: String

Operation Name: updateContacts

Figure 5.38: Add the operations to refresh the access token and to update the LinkedIn contacts.

Listing 5.31: The default configuration of the basic security filter in the Scout server's `config.ini` file.

```
### Servlet Filter Runtime Configuration
org.eclipse.scout.http.servletfilter.security.BasicSecurityFilter#active=↘
  →false
org.eclipse.scout.http.servletfilter.security.BasicSecurityFilter#realm=↘
  →contacts Development
org.eclipse.scout.http.servletfilter.security.BasicSecurityFilter#users=admin↘
  →\=manager,allen\=allen,blake\=blake
```

tained with `ServerSession.get().getUserId()`. Once we have both the access token and the user id available in method refreshToken, we can store the token and the secret values for the current user in table `USERS_PARAM` according to Listing 5.30.

In the current implementation of the "My Contacts" application this id will always be set to `anonymous`. The reason for this is the `AnonymousSecurityFilter` security filter that is activated during the initial project creation step. As for logging, and database access, the configuration of the security filters of the server application is defined in the server's `config.ini` file.

```
org.eclipse.scout.http.servletfilter.security.AnonymousSecurityFilter#active=↘
  →true
```

If you would like to change the "My Contacts" application to a multi-user setup, the next simpler option is to deactivate the anonymous security filter and use the basic security filter instead. In the server's `config.ini` file the necessary config lines are already prepared as shown in Listing 5.31. Just set `BasicSecurityFilter#active=true` and "My Contacts" the client application will prompt you with a login dialog. In this dialog you may enter one of the login credentials specified in in the `BasicSecurityFilter#users` parameter of the `config.ini` file. To log in as user admin, enter password "manager". To add new users, just amend the `BasicSecurityFilter#users` parameter with additional comma separated entries of the form
`username`
`p̄assword`.

Listing 5.32: The `updateContacts` method is used to enter/update existing contacts based on new data fetched from LinkedIn.

```java
@Override
public void updateContacts() throws ProcessingException {
  try {
    IPersonService service = SERVICES.getService(IPersonService.class);
    NodeList persons = readContacts();

    for (int i = 0; i < persons.getLength(); i++) {
      if (persons.item(i) instanceof Element) {
        Element person = (Element) persons.item(i);
        LOG.info(DomUtility.getString(person));

        // load existing person data
        PersonFormData formData = new PersonFormData();
        formData.setPersonId(DomUtility.getValue(person, "id"));
        service.load(formData);

        formData.getPictureUrl().setValue(
           DomUtility.getValue(person, "picture-url"));
        formData.getFirstName().setValue(
           DomUtility.getValue(person, "first-name"));
        formData.getLastName().setValue(
           DomUtility.getValue(person, "last-name"));
        formData.getHeadline().setValue(
           StringUtility.substring(DomUtility.getValue(person, "headline"), 0,↘
             → 64));
        formData.getLocation().setValue(
           DomUtility.getValue(person, "location", "name"));

        // save new/updated person data
        service.create(formData);
      }
    }
  }
  catch (Exception e) {
    throw new ProcessingException("LinkedIn Error", e);
  }
}
```

Listing 5.33: A sample contact represented in the XML format provided by the LinkedIn API.

```
<?xml version="1.0" encoding="UTF-16"?>
<person>
  <id>f7R6wGcblj</id>
  <first-name>Mike</first-name>
  <last-name>Milinkovich</last-name>
  <headline>Executive Director at Eclipse Foundation</headline>
  <picture-url>http://m3.licdn.com/mpr/mprx/0_IUM7Se9vBU...SbRbQZ4</picture-↘
    →url>
  <site-standard-profile-request>
   <url>http://www.linkedin.com/profile/view?id=14949387...05720*s114280*</↘
     →url>
  </site-standard-profile-request>
  <location>
   <name>Ottawa, Canada Area</name>
   <country>
    <code>ca</code>
   </country>
  </location>
  <industry>Computer Software</industry>
 </person>
```

The implementation of service method `updateContacts` of the LinkedIn service is provided in Listing 5.32. First, a service reference is obtained to the person process service. Then, the LinkedIn contacts obtained by method `readContacts` are stored in a `persons` list object. For each person, a form data is created and its person id parameter is set to `DomUtility.getValue(person, "id")`. Using the `load` method of the person process service all attributes of the person are loaded from the server's database. The new attribute values obtained from the corresponding LinkedIn contact are then used to update the corresponding form data parameters. The newly updated person form data is then stored in the database with `service.create(formData)`. Before we go through the `readContacts` method, we have a look at the explicit format of the contact data provided by the LinkedIn API.

From the sample XML extract provided in Listing 5.33, we can readily identify the person id `f7R6wGcblj`, the first name `Mike` and some other attribute names with associated attribute values. A closer look reveals the identity of the person to be Mike Milinkovich, a well known character in the Eclipse community. For each LinkedIn contact this format is parsed in method `readContacts` into a DOM tree[19]. Using the `DomUtility` helper class provided in Listing 5.34, these attribute values are then extracted from this DOM tree and used to update the existing person attributes.

Accessing the user's LinkedIn contact list is implemented in method `readContacts` according to Listing 5.35. In this method, a signed OAuth request is first created using the access token provided by method `getToken`. The request specified by constant `LINKEDIN_CONNECTIONS` is defined as `http://api.linkedin.com/v1/people/˜/connections`. According to the API specification[20] this returns all connections of the user. To sign the request, the access token is retrieved from the database of the "My Contact" application using method `getToken`. Ac-

---

[19] DOM stands for Document Object Model. See http://www.w3.org/DOM/ for more information.

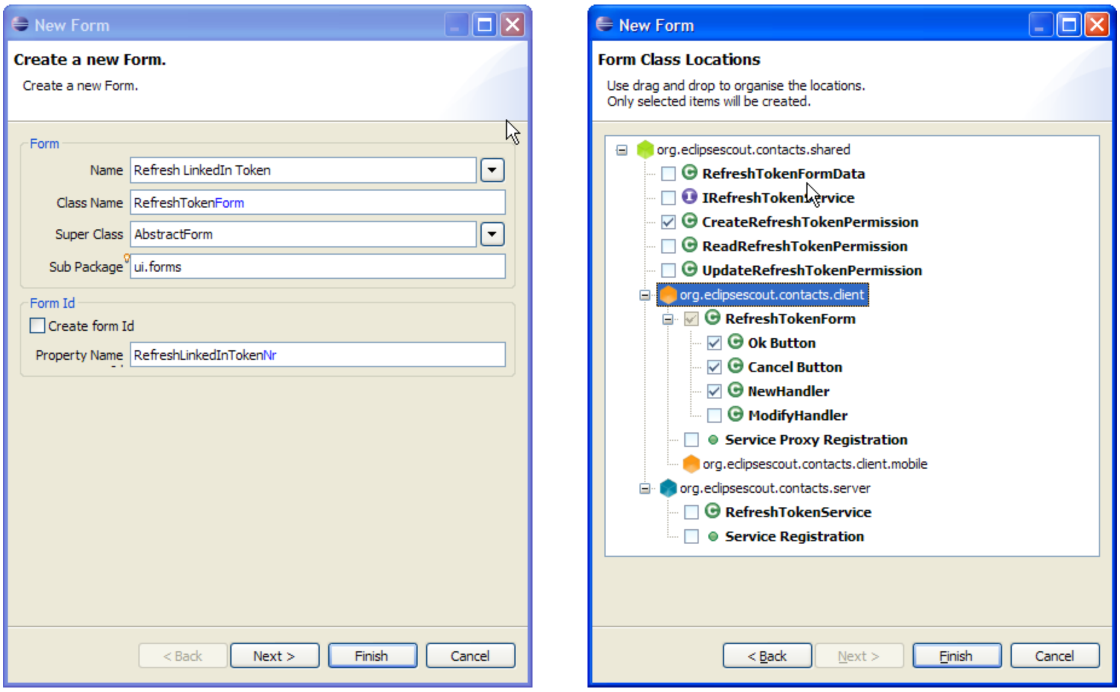
[20] See the LinkedIn API documented for more specific information: https://developer.linkedin.com/documents/linkedin-api-resource-map.

Listing 5.34: The `DomUtility` class provides functions to parse the XML data structure provided by the LinkedIn API.

```
public class DomUtility {
 private static Node getSubElement(Node parent, String name) {
  NodeList nodes = ((Element) parent).getElementsByTagName(name);
  if (nodes.getLength() == 0) {
   return null;
  }
  return nodes.item(0);
 }

 public static String getValue(Node node, String name) {
  Node n = getSubElement(node, name);
  if (n == null) {
   return "";
  }
  return n.getTextContent();
 }

 public static String getValue(Node node, String name, String subName) {
  Node n = getSubElement(node, name);
  if (n == null) {
   return "";
  }
  return getValue(n, subName);
 }

 public static String getString(Element element) {
  Document document = element.getOwnerDocument();
  DOMImplementationLS domImplLS = (DOMImplementationLS) document.↘
    →getImplementation();
  LSSerializer serializer = domImplLS.createLSSerializer();
  return serializer.writeToString(element);
 }
}
```

Listing 5.35: The `readContacts` method to fetch the users connection using the LinkedIn API. The necessary access token is created in method `getToken` based on the information stored in the database for the logged in user.

```
  private NodeList readContacts() throws Exception {
    // create singned linkedin request and get response
    OAuthRequest request = new OAuthRequest(Verb.GET, LINKEDIN_CONNECTIONS);
    m_service.signRequest(getToken(), request);
    Response response = request.send();

    // parse linkedin response stream
    DocumentBuilder builder = DocumentBuilderFactory.newInstance().↘
      →newDocumentBuilder();
    Document document = builder.parse(response.getStream());
    Element element = document.getDocumentElement();

    // basic error handling
    if (element.getNodeName().equals("error")) {
      LOG.error(DomUtility.getString(element));
      throw new ProcessingException(DomUtility.getString(element));
    }

    return element.getChildNodes();
  }

  private Token getToken() throws ProcessingException {
    String userName = ServerSession.get().getUserId();
    StringHolder value = new StringHolder();
    StringHolder secret = new StringHolder();

    SQL.selectInto("SELECT value INTO :value FROM USERS_PARAM WHERE param = :↘
      →param AND username = :username",
       new NVPair("value", value), new NVPair("param", LINKEDIN_TOKEN), new ↘
         →NVPair("username", userName));
    SQL.selectInto("SELECT value INTO :secret FROM USERS_PARAM WHERE param = :↘
      →param AND username = :username",
       new NVPair("secret", secret), new NVPair("param", LINKEDIN_SECRET), new↘
         → NVPair("username", userName));

    if (StringUtility.isNullOrEmpty(value.getValue())) {
      throw new ProcessingException("No valid LinkedIn token stored for user '"↘
        → + userName + "'. Please (re)create a token");
    }

    return new Token(value.getValue(), secret.getValue());
  }
}
```



Create a new Form.

Create a new Form.

Form

Name: Refresh LinkedIn Token

Class Name: RefreshTokenForm

Super Class: AbstractForm

Sub Package: ui.forms

Form Id

☐ Create form Id

Property Name: RefreshLinkedInTokenNr

Form Class Locations

Use drag and drop to organise the locations.
Only selected items will be created.

- org.eclipsescout.contacts.shared
  - ☐ RefreshTokenFormData
  - ☐ IRefreshTokenService
  - ☑ CreateRefreshTokenPermission
  - ☐ ReadRefreshTokenPermission
  - ☐ UpdateRefreshTokenPermission
  - org.eclipsescout.contacts.client
    - ☑ RefreshTokenForm
      - ☑ Ok Button
      - ☑ Cancel Button
      - ☑ NewHandler
      - ☐ ModifyHandler
    - ☐ Service Proxy Registration
    - org.eclipsescout.contacts.client.mobile
  - org.eclipsescout.contacts.server
    - ☐ RefreshTokenService
    - ☐ Service Registration

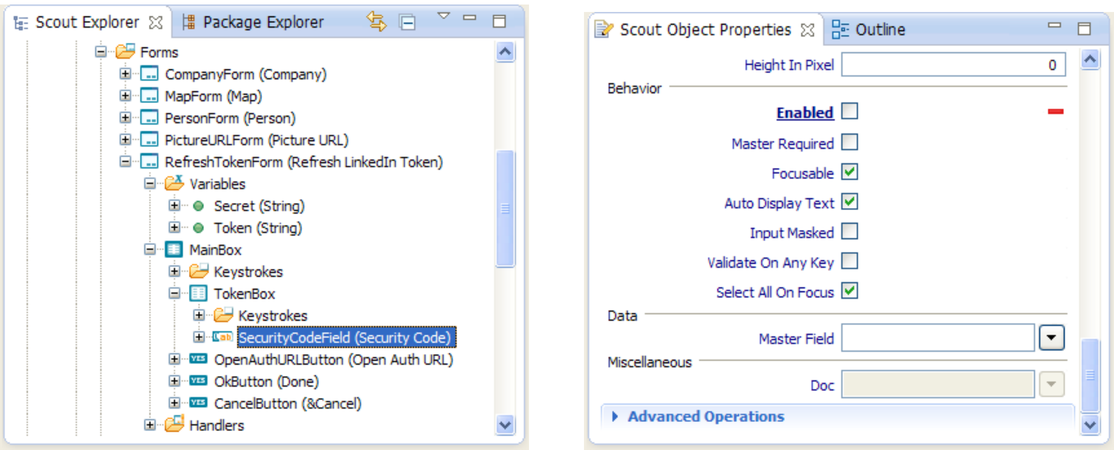
Figure 5.39: Add the form to refresh the LinkedIn access token.

cording to the implementation provided in Listing 5.35 the user id is first obtained from the user's server session. The necessary parameters to create the access token are then retrieved from the `USERS_PARAM` table. We have now implemented all necessary server services and operations to access the LinkedIn API, to create/refresh access tokens and to retrieve the users contacts.

5.12 Fetching Contacts from LinkedIn

The last piece that is missing to complete the "My Contacts" application is the user interaction for the LinkedIn integration implemented in the previous section. To allow the user to create/refresh an access token, we need to implement the refresh token form shown in Figure 5.2 at the beginning of this chapter. To create the form code, we use the *New Form* wizard of the Scout SDK as shown in Figure 5.39. For the *Name* field enter a new translated text "Refresh LinkedIn Token" and fill `ui.forms` into the *Sub Package* field field. Then, click the NEXT button to switch to the second wizard step. Here, deselect the elements that will not be needed for the implementation of the refresh form according to the right hand side of Figure 5.39. Once the form has been created by the wizard, add the a `Token` and a `Secret` form variable of type string under the form's *Variables* folder.

The next step is to add the desired layout and the necessary form fields. First, use the *New Form Field* wizard to add a new group box `TokenBox` to the *MainBox* node. And into the `TokenBox` add a "Security Code" string field as shown on the left hand side of Figure 5.40. To provide a click-able link that opens a web browser with the LinkedIn authentication link, add a "Open Auth URL" link of type `LinkButton` to the main box. To make the user to first click the authentication link button before he tries to fill any content into the security code field we initially



Height In Pixel: 0

Enabled ☐

Master Required ☐

Focusable ☑

Auto Display Text ☑

Input Masked ☐

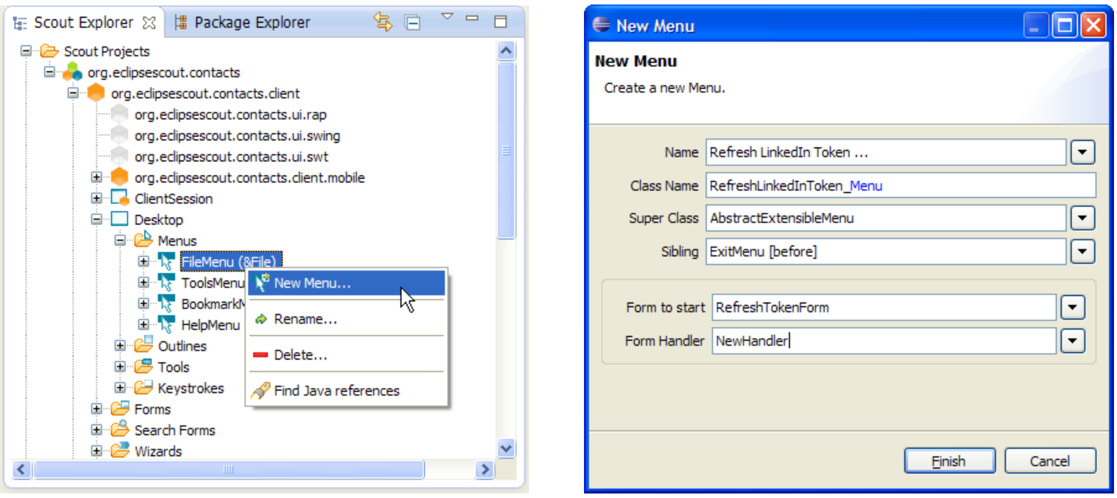
Validate On Any Key ☐

Select All On Focus ☑

Master Field:

Doc:

Figure 5.40: The token form in the explorer and the properties of the security code field.



New Menu

Create a new Menu.

Name: Refresh LinkedIn Token ...

Class Name: RefreshLinkedInToken_Menu

Super Class: AbstractExtensibleMenu

Sibling: ExitMenu [before]

Form to start: RefreshTokenForm

Form Handler: NewHandler

Figure 5.41: Add the menu to open the LinkedIn token form.

disable this field. For this, first click on the security code field in the Scout Explorer and then open the *Advanced Properties* section of the fields Scout Object Properties. As shown on the right side of Figure 5.40, deselect the *Enabled* property.

The implementation of the forms' structure is provided in Listing 5.36. In the `execClickAction` method of the link button `OpenAuthURLButton` the necessary information for the auth link is obtained from the server. For this, we are using the `getAuthUrl` operation implemented in the previous section. The parameters to create the request token are saved in the form's token and secret variables. Then, the auth link is opened in a web browser with the `shellOpen` method of the client's shell service. Finally, after opening the web browser we can enable the security field with `getSecurityCodeField().setEnabled(true)`.

We can now integrate the form to refresh the LinkedIn access token under the applications "File" menu. This is done using the *New Menu* wizard as shown in Figure 5.41. Enter the translated text "Refresh LinkedIn Token ..." into the wizard's *Name* field and pick the element

Listing 5.36: The structure of the refresh token form. The security code field gets enabled in method `execClickAction` after the authentication link button is pressed.

```
public class RefreshTokenForm extends AbstractForm {

 private String m_token;
 @Order(10.0)
 public class MainBox extends AbstractGroupBox {

  @Order(10.0)
  public class TokenBox extends AbstractGroupBox {

   @Order(10.0)
   public class SecurityCodeField extends AbstractStringField {

    @Override
    protected boolean getConfiguredEnabled() {
     return false;
    }

    @Override
    protected String getConfiguredLabel() {
     return TEXTS.get("SecurityCode");
    }
   }
  }

  @Order(20.0)
  public class OpenAuthURLButton extends AbstractLinkButton {

   @Override
   protected String getConfiguredLabel() {
    return TEXTS.get("OpenAuthURL");
   }

   @Override
   protected void execClickAction() throws ProcessingException {
    String[] auth = SERVICES.getService(ILinkedInService.class).getAuthUrl↘
     →();
    setToken(auth[0]);
    setSecret(auth[1]);

    SERVICES.getService(IShellService.class).shellOpen(auth[2]);
    getSecurityCodeField().setEnabled(true);
   }
  }
```

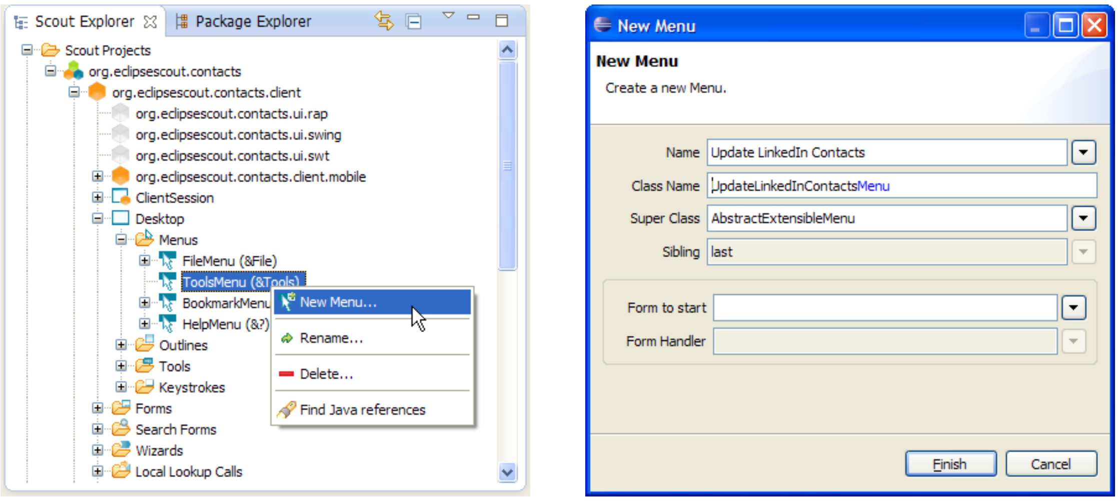
Listing 5.37: The menu to refresh the LinkedIn token starts the token form and then sends token parameters with the new security code to the LinkedIn backend service.

```
@Order(10.0)
public class RefreshLinkedInToken_Menu extends AbstractExtensibleMenu {

  @Override
  protected String getConfiguredText() {
    return TEXTS.get("RefreshLinkedInToken_");
  }

  @Override
  protected boolean getConfiguredEmptySpaceAction() {
    return true;
  }

  @Override
  protected void execAction() throws ProcessingException {
    RefreshTokenForm form = new RefreshTokenForm();
    form.startNew();
    form.waitFor();
    if (form.isFormStored()) {
      String token = form.getToken();
      String secret = form.getSecret();
      String securityCode = form.getSecurityCodeField().getValue();
      SERVICES.getService(ILinkedInService.class).refreshToken(token, secret↘
        →, securityCode);
    }
  }
}
```

Exit Menu [before] from the dropdown box of the *Sibling* field. In the *Form to start* field select the newly created refresh token form and use the *NewHandler* entry in the *Form Handler* field. To close the wizard, click the FINISH button.

The implementation of the refresh menu is shown in Listing 5.38. In method `execAction()`, we need to amend the part after starting the refresh token form. If the method `form.isFormStored()` returns `true`, the user has modified the secure code field and it is fair to assume that the user wants to create/refresh his LinkedIn access token. For this, we first retrieve the necessary parameters to call the backend service operation `refreshToken` implemented in the previous chapter.

As the last missing component of the "My Contacts" application, we add the menu to fetch the LinkedIn contacts and update the database accordingly. The menu entry is created as a sub menu of the "Tools" menu. For this, use the new menu wizard on the *ToolsMenu* node as shown in Figure 5.42. The implementation of the menu's `execAction` method shown in Listing 5.38 is trivial. We only need to call the operation of the `LinkedInService` implemented in the server part of the "My Contacts" application.



Name: Update LinkedIn Contacts
Class Name: UpdateLinkedInContactsMenu
Super Class: AbstractExtensibleMenu
Sibling: last

Figure 5.42: Add the menu to update the LinkedIn contacts.

Listing 5.38: The menu to update the stored persons with current LinkedIn data.

```
@Order(10.0)
public class UpdateLinkedInContactsMenu_ extends AbstractExtensibleMenu {

  @Override
  protected String getConfiguredText() {
    return TEXTS.get("UpdateLinkedInContacts");
  }

  @Override
  protected void execAction() throws ProcessingException {
    Desktop.this.setStatusText(TEXTS.get("ImportingLinkedIn"));
    SERVICES.getService(ILinkedInService.class).updateContacts();
    Desktop.this.setStatusText(null);
  }
}
```

Part II

Appendices

Appendix A

Licence and Copyright

This appendix first provides a summary of the Creative Commons (CC-BY) licence used for this book. The licence is followed by the complete list of the contributing individuals, and the full licence text.

A.1 Licence Summary

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit https://creativecommons.org/licenses/by/3.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

A summary of the license is given below, followed by the full legal text.

You are free:

- **to Share**—to copy, distribute and transmit the work
- **to Remix**—to adapt the work
- to make commercial use of the work

Under the following conditions:

Attribution—You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

Waiver—Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain—Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights—In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice—For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to `https://creativecommons.org/licenses/by/3.0/`.

A.2 Contributing Individuals

Copyright (c) 2012-2013.

In the text below, all contributing individuals are listed in alphabetical order by name. For contributions in the form of GitHub pull requests, the name should match the one provided in the corresponding public profile.

Matthias Zimmermann Alex Schroeder

A.3 Full Licence Text

The full licence text is availble online at `http://creativecommons.org/licenses/by/3.0/legalcode`

```
Creative Commons
Attribution 3.0 Unported

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE
LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN
ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS
INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES
REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR
DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS
PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE
WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS
PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND
AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS
LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU
THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH
```

TERMS AND CONDITIONS.

1. Definitions

   a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

   b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

   c. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

   d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

   e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

   f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

   g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

   h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

   i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

   b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

   c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

   d. to Distribute and Publicly Perform Adaptations.

   e. For the avoidance of doubt:

      i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

      ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

      iii. Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(b), as requested.

   b. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor’s copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by

Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

c. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE

WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

Appendix B

Scout Installation

B.1 Overview

This chapter walks you through the installation of Eclipse Scout. The installation description (as well as the rest of this book) is written and tested for Eclipse Scout 3.9. Scout 3.9.0 will be delivered as part of the Eclipse Kepler release train on June 26, 2013. Detailed information regarding the scheduling of the Kepler release train is provided in the Eclipse wiki[1].

We assume that you have not installed any software relevant for the content of this book. This is why the Scout installation chapter starts with the installation of the Java Development Kit (JDK). Consequently, you will have to skip some of the sections depending on your existing setup.

In the text below, installation routines are described separately for Windows, Mac, and Linux. As Scout applications have been built primarily on the Windows platform in the past, Scout also has the highest maturity level on this platform.

B.2 Download and Install a JDK

The first step to install Scout is to have an existing and working installation of a JDK. It is recommended to go for the most recent download of Java 7.

You may also use Scout with Java 6. However, be aware that the communicated end of life for Oracle's Java 6 is February 2013.[2] Java 5 is not supported. Running Scout on Java 8 is realistic for the 2014 release train of Eclipse. This guess is based on the Java SE Roadmap press release.[3]

B.2.1 Windows

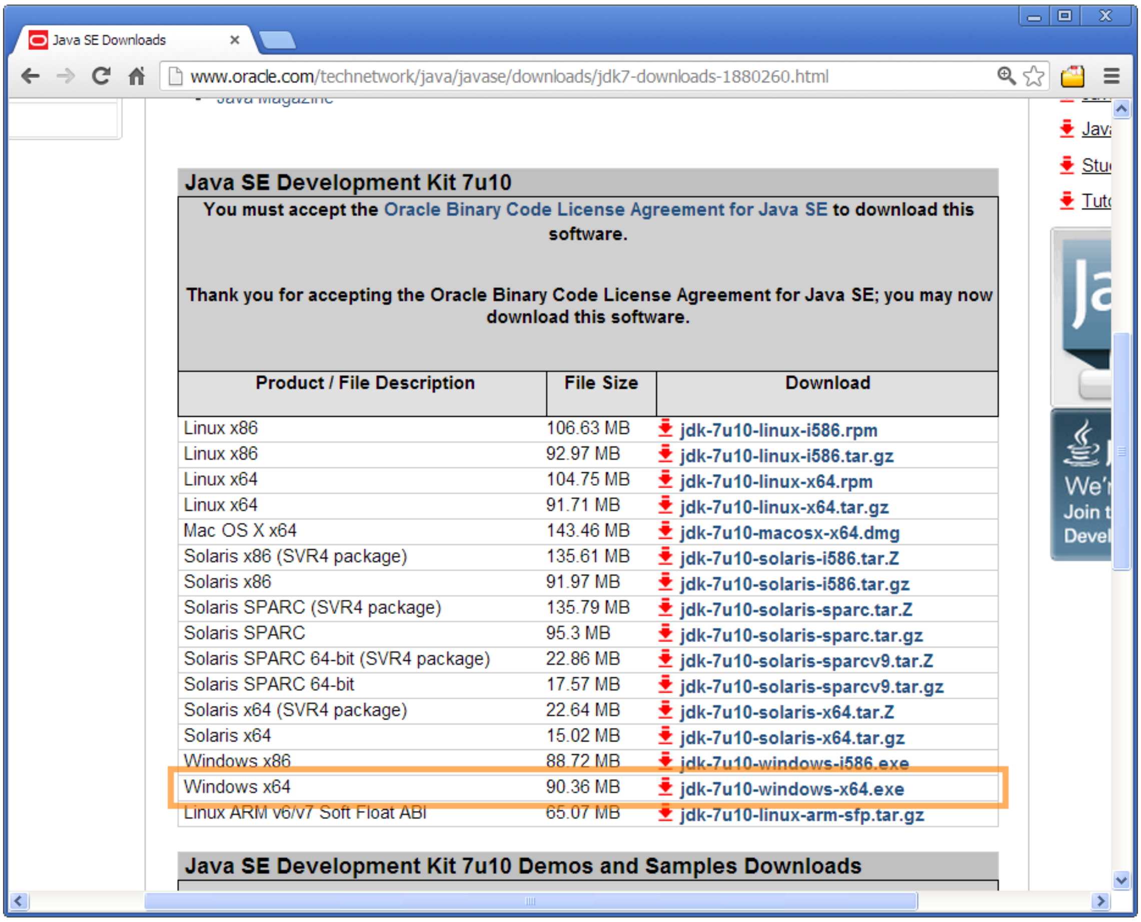
On the Windows platform, we recommend to install the Oracle JDK 7 together with Scout. The reason for this recommendation is simple. Eclipse Scout is officially tested with the Oracle JDK 7 on Windows.

---

[1] Kepler release plan: `http://wiki.eclipse.org/Kepler/Simultaneous_Release_Plan`

[2] Java 6 End of Life: `http://www.oracle.com/technetwork/java/eol-135779.html`

[3] Java SE Roadmap press release: `http://www.oracle.com/us/corporate/press/1854982`



Java SE Downloads
www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

Java SE Development Kit 7u10

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	106.63 MB	jdk-7u10-linux-i586.rpm
Linux x86	92.97 MB	jdk-7u10-linux-i586.tar.gz
Linux x64	104.75 MB	jdk-7u10-linux-x64.rpm
Linux x64	91.71 MB	jdk-7u10-linux-x64.tar.gz
Mac OS X x64	143.46 MB	jdk-7u10-macosx-x64.dmg
Solaris x86 (SVR4 package)	135.61 MB	jdk-7u10-solaris-i586.tar.Z
Solaris x86	91.97 MB	jdk-7u10-solaris-i586.tar.gz
Solaris SPARC (SVR4 package)	135.79 MB	jdk-7u10-solaris-sparc.tar.Z
Solaris SPARC	95.3 MB	jdk-7u10-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	22.86 MB	jdk-7u10-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.57 MB	jdk-7u10-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	22.64 MB	jdk-7u10-solaris-x64.tar.Z
Solaris x64	15.02 MB	jdk-7u10-solaris-x64.tar.gz
Windows x86	88.72 MB	jdk-7u10-windows-i586.exe
Windows x64	90.36 MB	jdk-7u10-windows-x64.exe
Linux ARM v6/v7 Soft Float ABI	65.07 MB	jdk-7u10-linux-arm-sfp.tar.gz

Java SE Development Kit 7u10 Demos and Samples Downloads

Figure B.1: Installer download for Oracle JDK 7. The Windows 64bit installer package is highlighted.

To successfully install the JDK you need to have at least local admin rights. You also need to know your hardware architecture in order to download the correct JDK installer. The steps guide[6]. To verify the installation you might want to go through this Java "Hello World!" tutorial[7].

B.3 Download and Install Scout

Installing Eclipse Scout 3.9 requires a working JDK 6 or JDK 7 installation. If this is missing, see the previous section for installation instructions.

The download page should look as shown in Figure B.2. In this screenshot, the packages available for *Eclipse Kepler (4.3) M4* are shown and *Windows* is preselected in the dropdown list. Should the download page show the wrong platform in the dropdown list, you will need to choose the correct one manually.

The following sections describe the installation steps depending on the operating system of your environment.

B.3.1 Windows

For Windows, the Eclipse download page offers two versions for each package. In Figure B.2, the links for these two versions are called *Windows 32 Bit* and *Windows 64 Bit*. To decide which version to download you first need to know if you have installed a 64-bit JDK or a 32-bit JDK and then download the corresponding Eclipse installation. Because this is important, let us be very explicit: **If you are using a 64-bit JDK you must download the 64-bit version of the Eclipse Scout package. If you are using a 32-bit JDK you must download the 32-bit version of the Eclipse Scout package.** Any other combination will simply not work.

If you just installed the JDK yourself you will remember what you have installed. In case you have a JDK pre-installed you can find out by typing `java -version` on your console. For 64-bit JDK installations this fact is explicitly mentioned in the text printed by this command. The output will then look similar to the one provided below.

```
C:\Users\scouty>java -version
java version "1.7.0_02"
Java(TM) SE Runtime Environment (build 1.7.0_02-b13)
Java HotSpot(TM) 64-Bit Server VM (build 22.0-b10, mixed mode)
```

If you have installed a 32-bit version this fact is not explicitly mentioned. See below for an example output of a 32-bit installation.

```
C:\Users\scouty>java -version
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode, sharing)
```

Now you know which version of the Eclipse Scout package to download. Installation of the Scout package does not require any administrative priviledges, unlike the JDK installation. As the package is a simple ZIP file, you may unpack its content to a location of your choice as shown in Figure B.3. If possible, we recommend a target directory such as `C:\scout\kepler_m4`. Such

---

[6] Install the JDK: http://docs.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html#Run

[7] Windows Java "Hello World!": http://docs.oracle.com/javase/tutorial/getStarted/cupojava/win32.html

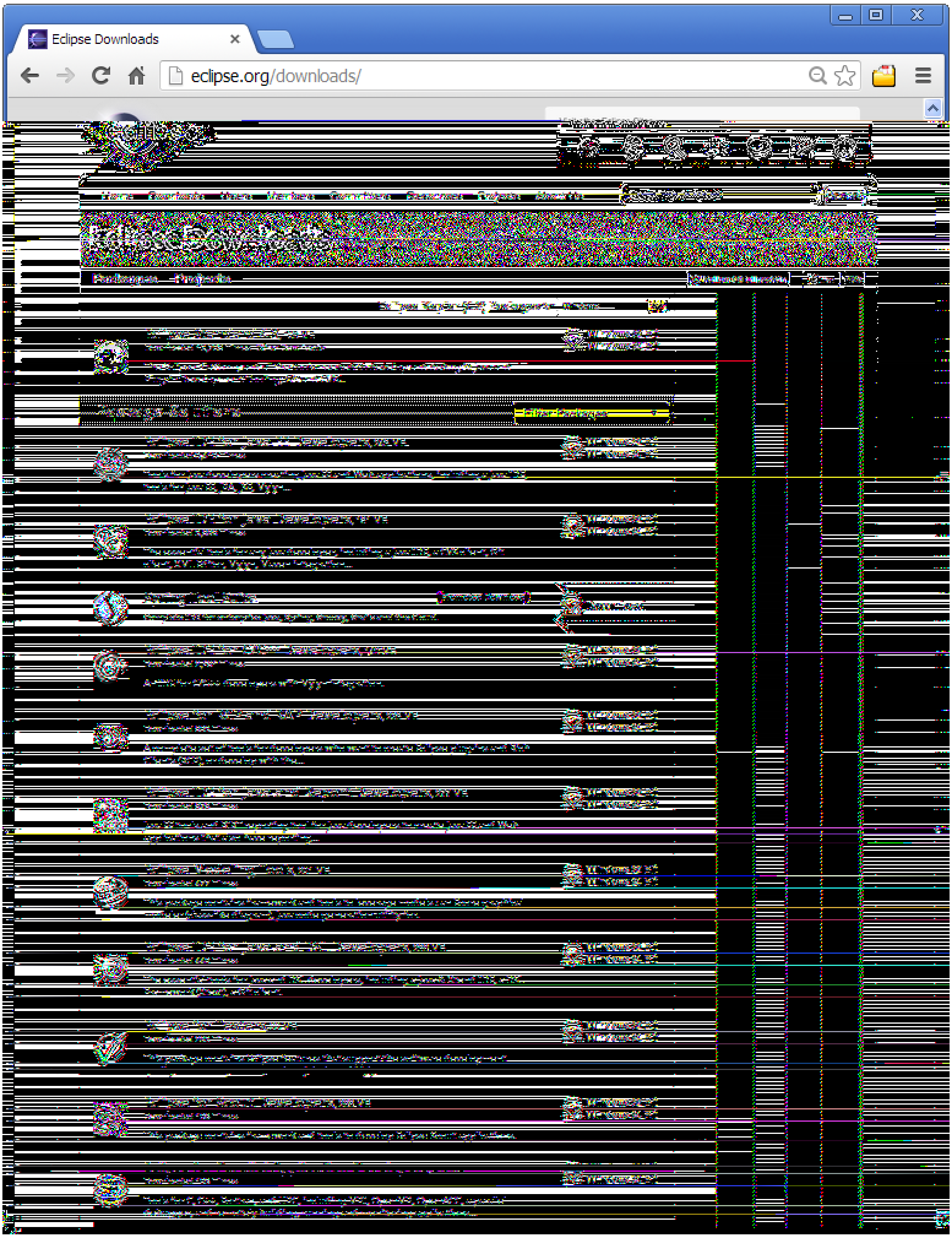
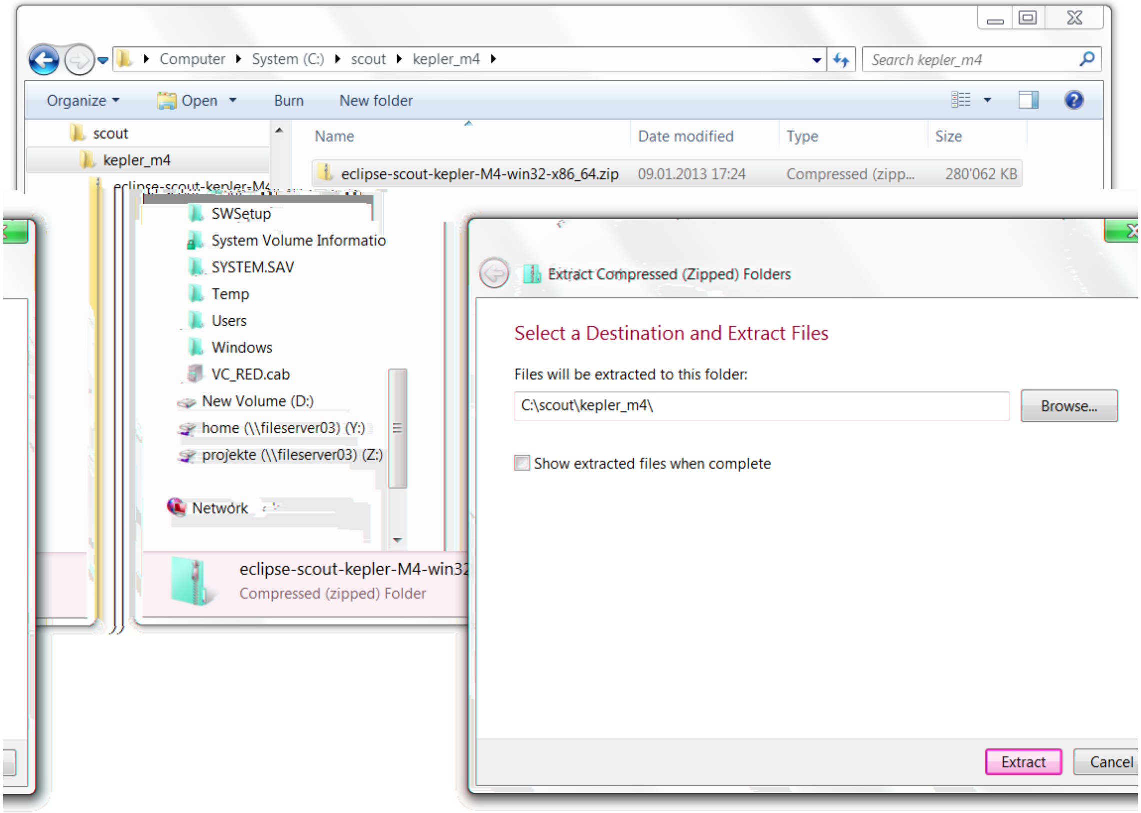


Figure B.2: Eclipse Kepler developer builds download page. The Scout package is highlighted



Name	Date modified	Type	Size
eclipse-scout-kepler-M4-win32-x86_64.zip	09.01.2013 17:24	Compressed (zipp...	280'062 KB

Select a Destination and Extract Files

Files will be extracted to this folder:

C:\scout\kepler_m4\

☐ Show extracted files when complete

Figure B.3: Extracting Eclipse Scout to its target directory

a target directory avoids two problems that we had to deal with in the past: path names including spaces, and very long path names that may break the functionality of your zip/unzip tool.

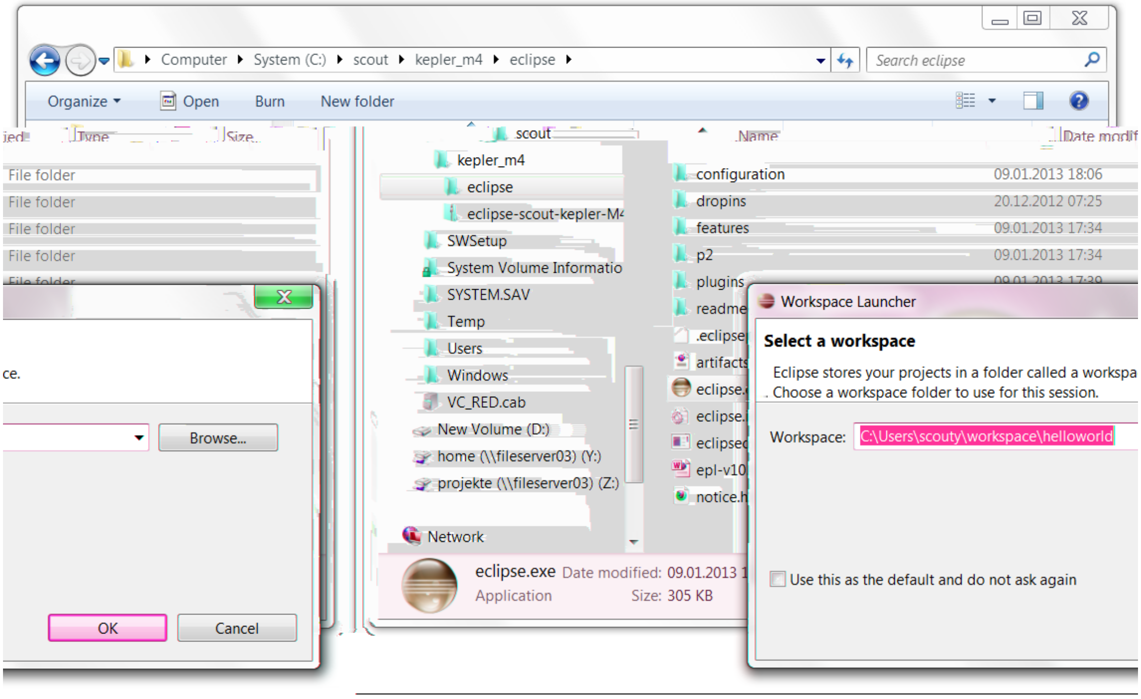
After unzipping the Scout package, navigate into the subfolder `Eclipse`. In case you have installed multiple JDKs coming with their individual Java Runtime Environments (JREs), you might want to explicitly specifiy which JRE to use. Open the file `eclipse.ini` in a editor of your choice and insert the following two lines at the top of the file:

```
-vm
C:\java\jre7\bin\javaw.exe
```

where the second line specifies the exact path to the JRE to be used to start your Eclipse Scout installation. If you have only installed a single JDK you will not need to change the default `eclipse.ini` file of your Eclipse installation.

You are now ready to start Eclipse Scout via the `eclipse.exe` file. This brings up the workspace launcher as shown in Figure B.4. Into the *Workspace* field you enter the target directory of your Scout project. For a new project you can also enter a path to a directory that does not yet exist. After clicking the Ok button, Eclipse will create directories that do not yet exist and start the workspace from the specified location.

When opening a new workspace for the first time you will be shown the welcome screen as shown in Figure B.5. Click on the WORKBENCH icon to open the Eclipse Scout perspective as



Select a workspace

Eclipse stores your projects in a folder called a workspa
. Choose a workspace folder to use for this session.

Workspace: C:\Users\scouty\workspace\helloworld

☐ Use this as the default and do not ask again

Figure B.4: Starting the Eclipse Scout package.

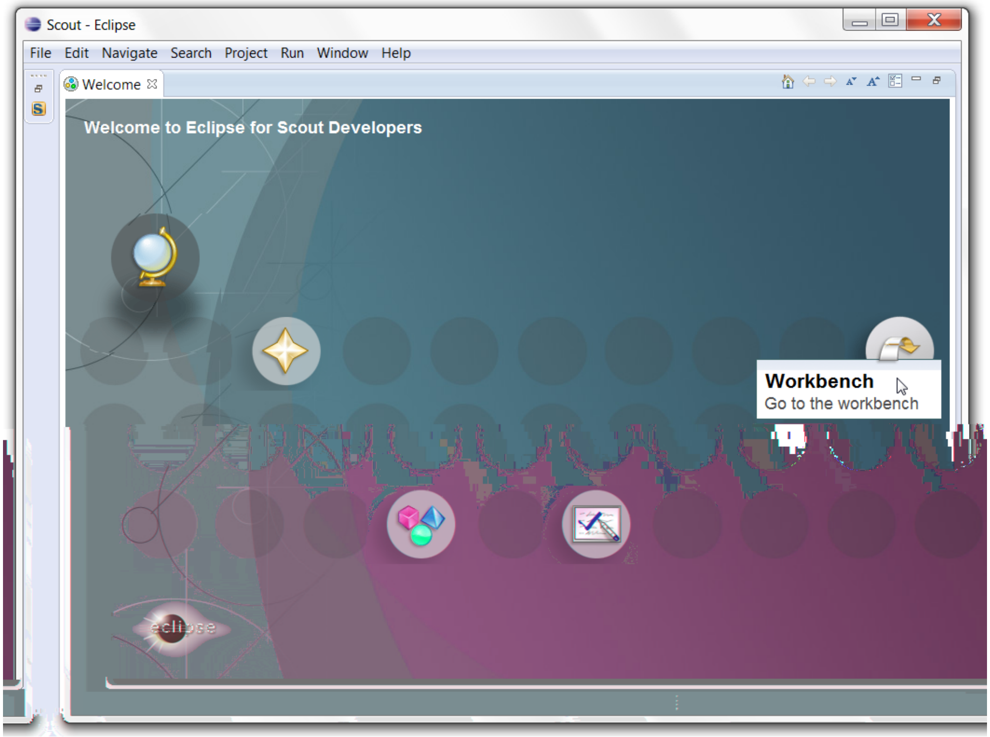

Scout - Eclipse
File Edit Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse for Scout Developers
Workbench
Go to the workbench
eclipse

Figure B.5: Eclipse Scout welcome screen.

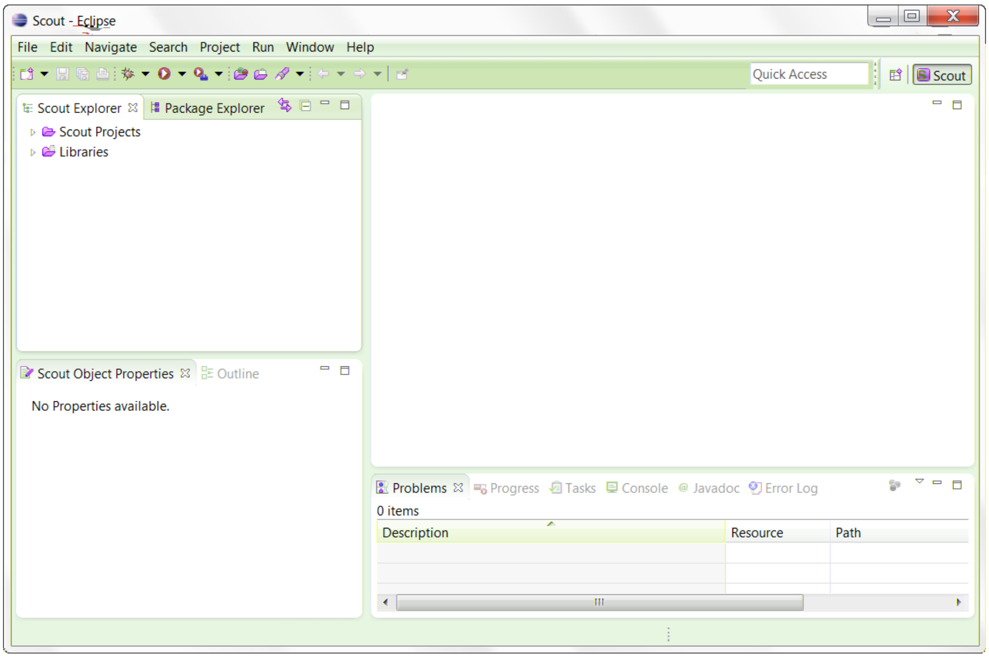


Figure B.6: Eclipse Scout started in the Scout SDK perspective.

shown in Figure B.6.

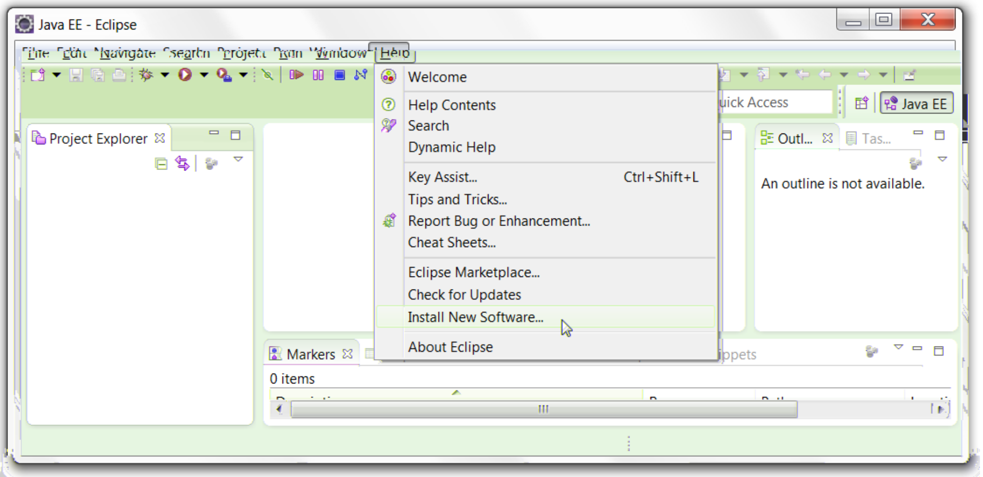
If you have explicitly specified the JRE to be used you verify this in the running Eclipse installation. Fist, select the Help—About Eclipse menu to open the about dialog. Then, click on the Installation Details button and switch to the Configuration tab. In the long list of system properties you will find lines similar to the ones shown below.

```
*** Date: Mittwoch, 9. Januar 2013 20:05:53 Mitteleuropische Normalzeit

*** Platform Details:

*** System properties:
...
-vm
C:\java\jre7\bin\javaw.exe
...
sun.java.command=... vm C:\java\jre7\bin\javaw.exe -vmargs ...
```

You have now successfully completed the Eclipse Scout installation on your Windows environment. With this running Scout installation you may skip the following section on how to add Scout to an existing Eclipse installation.

B.4 Add Scout to your Eclipse Installation

This section describes the installation of Scout into an existing Eclipse installation. As the audience of this section is assumed to be familiar with Eclipse, we do not describe how you got your Eclipse



An outline is not available.

Figure B.7: Eclipse menu to install additional software

installation in the first place. For the provided screenshots we start from the popular package *Eclipse IDE for Java EE Developers* from the Juno release.

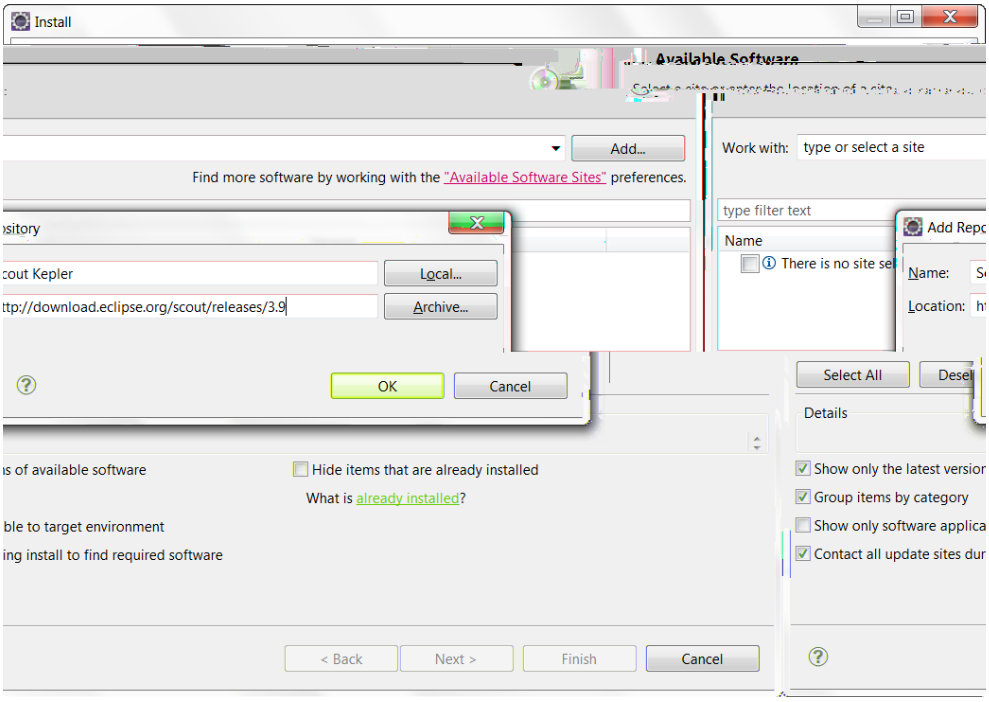
To add Scout to your existing Eclipse installation, you need to start Eclipse. Then select the HELP—INSTALL NEW SOFTWARE... menu as shown in Figure B.7 to open the install dialog.

In the install dialog, click on the ADD... button to enter the link to the Scout repository. This opens the popup dialog *Add Repository* As shown in Figure B.8 you may use ”Scout Kepler” for the *Name* field. For the *Location* field enter the Scout Kepler release repository as specified below. `http://download.eclipse.org/scout/releases/3.9`.

After connecting to the Scout Kepler repository you may both Scout features as shown in Figure B.9. Then, move through the installation with the NEXT button and accept the presented EPL terms. Currently, there is still unsigned content included in the features. So please click away the corresponding warning and accept the request for a restart of Eclipse. Finally, you may add the Scout perspective to end up at a state very similar to Figure B.6.

B.5 Verifying the Installation

After you can start your Eclipse Scout package you need to verify that Scout is working as intended. The simplest way to verify your Scout installation is to create a ”Hello World” Scout project and run the corresponding Scout application as described in Chapter 2.



Find more software by working with the "Available Software Sites" preferences.

ository

cout Kepler

ttp://download.eclipse.org/scout/releases/3.9

Work with: type or select a site

type filter text

☐ There is no site sel

Name: S

Location: h

s of available software

ble to target environment

ing install to find required software

☐ Hide items that are already installed

What is already installed?

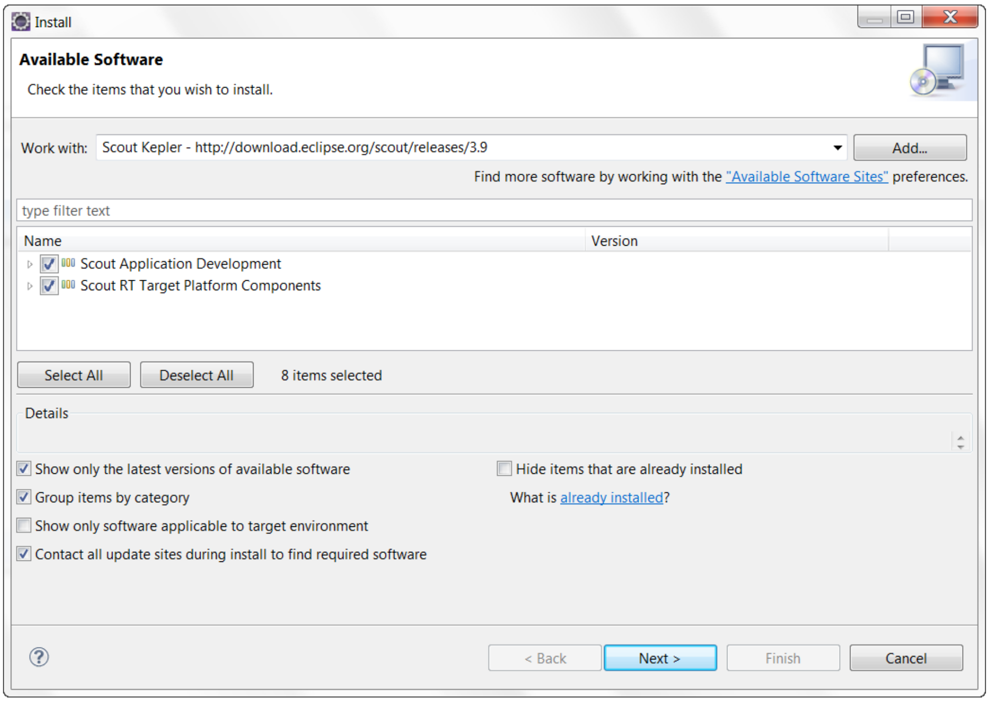
☑ Show only the latest versior

☑ Group items by category

☐ Show only software applica

☑ Contact all update sites dur

Figure B.8: Add the Scout Kepler repository



Available Software

Check the items that you wish to install.

Work with: Scout Kepler - http://download.eclipse.org/scout/releases/3.9

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☑ Scout Application Development	
☑ Scout RT Target Platform Components	

8 items selected

☑ Show only the latest versions of available software

☑ Group items by category

☐ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

☐ Hide items that are already installed

What is already installed?

Figure B.9: Select the Scout features to add to the Eclipse installation

Appendix C

Apache Tomcat Installation

Apache Tomcat is an open source web server that is a widely used implementation of the Java Servlet Specification. Specifically, Tomcat works very well to run the server part of Scout client server applications. In case you are interested in getting some general context around Tomcat you could start with the Wikipedia article[1]. Then get introduced to its core component "Tomcat Catalina"[2] before you switch to the official Tomcat homepage[3].

This section is not really a step by step download and installation guide. Rather, it points you to the proper places for downloading and installing Tomcat. We recommend to work with Tomcat version 7.0 but in case you already have version 6.0 running that should be good too. Start your download from the official download site[4].

Once you have downloaded and installed Tomcat 7 (see the sections below for plattform specific guidelines) you can start the corresponding service or deamon. To verify that Tomcat is actually running open a web browser of your choice and type `http://localhost:8080` into the address bar. You should then see a confirmation of the successful installation according to Figure C.1.

C.1 Windows Installation

According to the Tomcat setup installation for Windows[5] download the package "32-bit/64-bit Windows Service Installer" from the Tomcat 7 download site. Then, start the installer and accept the proposed default settings.

For installing Tomcat on OS X systems download the "tar.gz" package from the Tomcat 7 download site. Then, follow the installation guide[6] provided by Wolf Paulus.

For Linux systems download the "tar.gz" package from the Tomcat 7 download site. Then, follow the description of the Unix setup[7] to run Tomcat as a deamon. If you use Ubuntu, you may

---

[1] Apache Tomcat Wikipedia: `http://en.wikipedia.org/wiki/Apache_Tomcat`.

[2] Mulesoft's introduction to Tomcat Catalina: `http://www.mulesoft.com/tomcat-catalina`.

[3] Apache Tomcat Homepage: `http://tomcat.apache.org/`

[4] Tomcat 7 Downloads: `http://tomcat.apache.org/download-70.cgi`

[5] Tomcat Windows setup: `http://tomcat.apache.org/tomcat-7.0-doc/setup.html#Windows`

[6] Installing Tomcat on OS X: `http://wolfpaulus.com/journal/mac/tomcat7`

[7] Tomcat Linux setup: `http://tomcat.apache.org/tomcat-7.0-doc/setup.html#Unix_daemon`

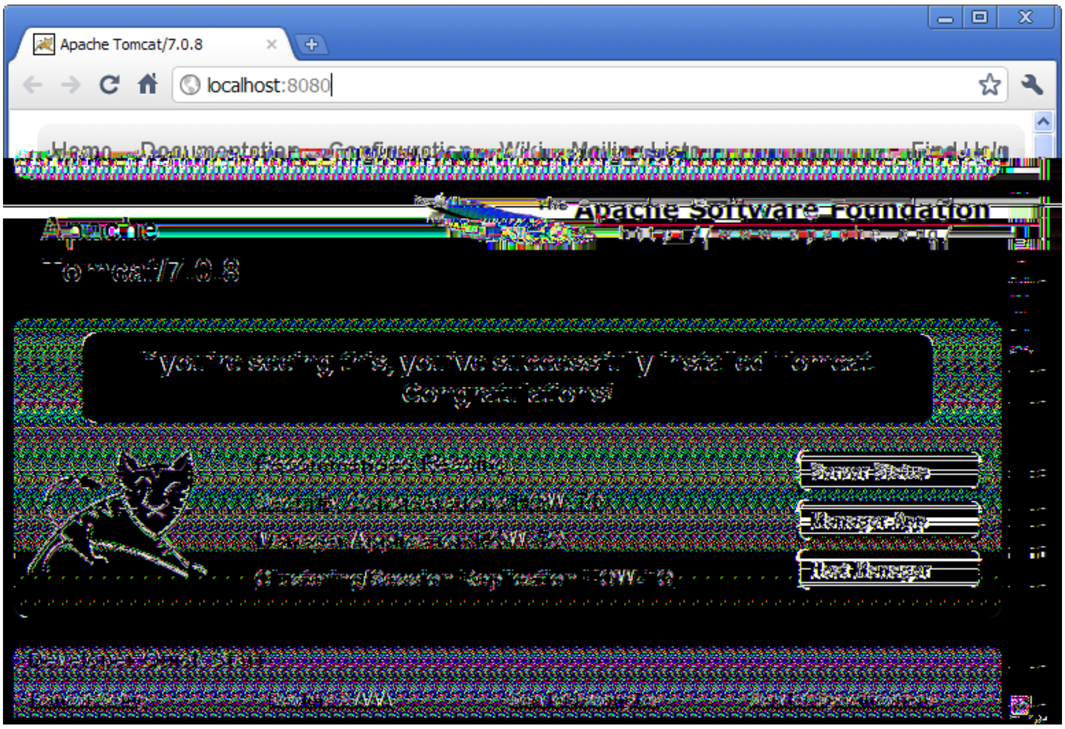


Figure C.1: A successful Tomcat 7 installation.

want to follow the tutorial[8] for downloading and installing Tomcat provided by Lars Vogel.

C.2 Directories and Files

Tomcat's installation directory follows the same organisation on all platforms. Here, we will only introduce the most important aspects of the Tomcat installation for the purpose of this book.

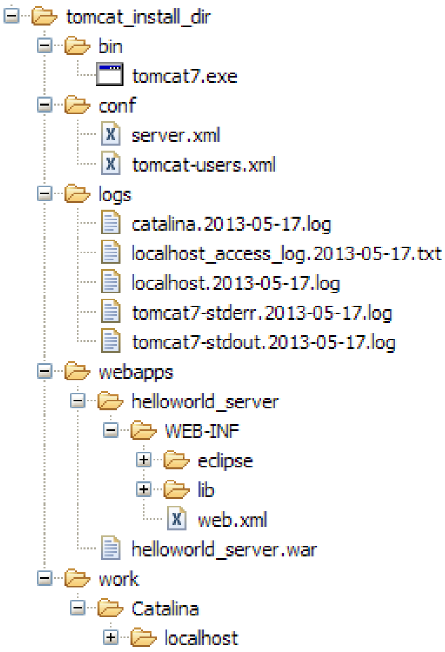
Note that some folders and many files of a Tomcat installation are not represented in Figure C.2. We just want to provide a basic understanding of the most important parts to operate the web server in the context of this book. In the `bin` folder, the executable programs are contained, including scripts to start and stop the Tomcat instance.

The `conf` folder contains a set of XML and property configuration file. The file `server.xml` represents Tomcat's main configuration file. It is used to configure general web server aspects such as the port number of its connectors for the client server communication. For the default setup, port number 8080 is used for the communication between clients applications and the web server. The file `tomcat-users.xml` contains a database of users, passwords and associated roles. Folder `logs` contains various logfiles of Tomcat itself as well as host and web application log files.

The folder needed for deploying web applications into a Tomcat instance is called `webapps`. It can be used as the target for copying WAR files into the web server. The installation of the WAR file then extracts its content into the corresponding directory structure as shown in Figure C.2 in

---

[8] Apache Tomcat Tutorial: `http://www.vogella.com/articles/ApacheTomcat/article.html`



```
tomcat_install_dir
    bin
        tomcat7.exe
    conf
        server.xml
        tomcat-users.xml
    logs
        catalina.2013-05-17.log
        localhost_access_log.2013-05-17.txt
        localhost.2013-05-17.log
        tomcat7-stderr.2013-05-17.log
        tomcat7-stdout.2013-05-17.log
    webapps
        helloworld_server
            WEB-INF
                eclipse
                lib
                web.xml
        helloworld_server.war
    work
        Catalina
            localhost
```

Figure C.2: The organisation of a Tomcat installation including specific files of interest. As an example, the "Hello World" server application is contained in subdirectory `webapps`.

the case of the file `helloworld_server.war`.

Finally, folder `work` contains Tomcat's runtime "cache" for the deployed web applications. It is organized according to the hierarchy of the engine (Catalina), the host (localhost), and the web application (helloworld$_{s}erver$).

C.3 The Tomcat Manager Application

Tomcat comes with the pre installed "Manager App". This application is useful to manage web applications and perform tasks such as deploying a web application from a WAR file, or starting and stopping installed web applications. A comprehensive documentation for the "Manager App" can be found under the Tomcat homepage[9]. Here we only show how to start this application from the hompage of a running Tomcat installation.

To access this application you can switch to the "Manager App" with a click on the corresponding button on the right hand side. The button can be found on the right hand side of Figure C.1. Before you are allowed to start this application, you need to provide username and password credentials of a user associated with Tomcats's `manager-gui` role.

To get at user names and passwords you can open file `tomcat-users.xml` located in Tomcat's `conf` directory. In this file the active users with their passwords and associated roles are stored. See Listing C.1 for an example. From the content of this file, we see that user `admin` has password `s3cret` and also posesses the necessary role `manager-gui` to access the "Man-

---

[9] The Tomcat Manager Application: `http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html`.

Listin n0 F678.9664 58.89320 e 43 1 n 28ten t 43 r 43 0

6 1s 6g g 1sg see B 6 e s 6 e

Bibliography

[1] *Bert Bates, Kathy Sierra*, **Head First Java** 2nd edition, O'Reilly Media, 2005.

[2] *Joshua Bloch*, **Effective Java** 2nd edition, Addison-Wesley, 2008.

[3] *Bruce Eckel*, **Thinking in Java** 4th edition, Prentice Hall International, 2006.

Index

Symbols

AbstractFormHandler 107
AbstractLinkButton 121
BasicSecurityFilter 114
CompanyLookupCall 101
DesktopForm 34
DesktopFormData 23
IOUtility 96
IShellService 79
LookupCall 101
MainBox 42
ModifyHandler 107
NVPair 92
PersonService 108 f.
SQL 92
ScoutXlsxSpreadsheetAdapter 79
ServerApplication 90
ServerSession 90
StandardOutlineService 92
StringUtility 92
UserAgentUtility 80
ViewHandler 36
execAction 99
execChangedMasterValue 96
execClickAction 107, 121
execCreateChildPages 81
execInitField 106
execOpened 80
execRowAction 100
getConfiguredLookupCall 105
getConfiguredMasterField 96
getConfiguredService 101
initializeService 112
isFormStored 100
reloadPage 100
shellOpen 79

A

Adding application languages 69

E

Eclipse config.ini file 39
Eclipse IDE 53
Eclipse plugin editor 45
Eclipse product file editor 39

M

Managing translated application texts 69

N

NLS editor 43, 69

P

Product launchers 18
Project creation 15

R

RAP 41

S

Scout Explorer 56
Scout Object Properties 59
Scout SDK 54
SDK Wizard 62
  Add Translation Entry 20
  Export Scout Project 25, 64
  New Form Field 20, 66
  New Scout Project 17, 62