Developing Fabasoft Folio Cloud Apps

Andreas Hofmann

Fabasoft®

Fabasoft®

ISBN: 978-3-902495-28-0

Honauerstraße 4, 4020 Linz
Tel.: +43 (732) 606162
http://*www.fabasoft.com*

Developing Fabasoft Folio Cloud Apps

Andreas Hofmann

Contents

Getting started

1 Getting started

1.1 What is Fabasoft Folio Cloud?

Fabasoft Folio Cloud is a public cloud service for efficient and secure online collaboration. The software implements internationally recognized security standards such as ISO27001 and SAS 70 Type II to guarantee the highest security standards and reliability, and features an intuitive user interface, which is available in 16 different languages.

It is based on a compelling freemium model that brings you the Fabasoft Folio Cloud primo edition with all the features needed for efficient online collaboration entirely free of charge while the premium editions with advanced features and functionality come at a charge.

With Fabasoft Folio Cloud, you can access your data from anywhere in the world and at any time you want, create new documents or upload files and securely share them with other people.

Protecting your confidential online data is of the utmost importance to us, and thanks to numerous security features, encrypted data transfer and sophisticated authentication mechanisms, Fabasoft Folio Cloud guarantees the highest safety and security levels for all your online activities.

Fabasoft Folio Cloud also offers some of the simplest and easiest ways of collaborating online. Within minutes you can set up your teams, organize your project tasks and manage shared contacts, appointments and documents. Team rooms and other collaboration features facilitate your online collaboration experience – within your company as well as with international business partners, or with friends at home and abroad.

In addition, Fabasoft Folio Cloud provides a robust and secure platform for fully fledged professional business applications, supporting thousands of users via nifty, exciting and intuitive Cloud Apps that make your online and mobile life easier and more enjoyable.

… And this is where you come into play: the Cloud App developer!

With Fabasoft app.ducx, Fabasoft app.test and Fabasoft app.telemetry you can easily and rapidly create your own Cloud Apps, bring them online in Fabasoft Folio Cloud and make them available to thousands of users!

1.2 What is Fabasoft app.ducx?

Fabasoft app.ducx is the agile, use case-oriented development platform for Cloud Apps. It has been specifically designed to cover all your needs when developing Cloud Apps and supports you throughout the entire software development life cycle. The efficient implementation of Cloud Apps is facilitated by domain-specific languages.

Cloud App development requires managing different aspects and elements such as data structures, user interface design, the implementation of methods and business rules. In order to account for this concept in an optimal manner, Fabasoft app.ducx is comprised of several declarative modeling languages, each designed for covering a particular aspect of solution development.

For example, Fabasoft app.ducx includes a modeling language that has been designed explicitly for the definition of an object model. In addition to this, Fabasoft app.ducx includes languages for defining resources, a user interface model, an implementation model, and a process model.

These modeling languages are referred to as domain-specific languages (DSLs), where each DSL was designed for addressing a certain aspect of Cloud App development.

1.3 Who should read this book?

The answer is simple enough: You should read this book!

At Fabasoft, we continuously strive to make Cloud App development as simple as humanly possible, because we believe that anybody should be given the tools to be able to create vivid apps from great ideas in a simple, rapid, agile and enjoyable way.

Even though this book is aimed primarily at software developers interested in exploring the tremendous potential of Fabasoft Folio Cloud, you do not need to be a professional programmer to create your own Cloud Apps.

As will be shown in the following chapters, all you need to get started is a basic understanding of a few technologies related to web development and object-oriented programming.

This book assumes that you have some level of familiarity with web technologies, object-oriented programming and Eclipse. Reading this book in conjunction with other books that are devoted specifically to these topics may be useful if you are not already comfortable using these technologies. Furthermore, the bibliography on page 175 provides some helpful resources.

For your convenience, concepts and technologies specific to Fabasoft Folio Cloud development are explained in great detail throughout the book.

1.4 Why develop Cloud Apps?

Now that you have an understanding of the key features of Fabasoft Folio Cloud, you will probably ask: How can I contribute, and what's in it for me?

Well, at Fabasoft we want to make it worth your while to put your creativity and time into developing Cloud Apps. A revenue sharing model described in "Reaping the profits" on page 159 allows you to profit from the success of Fabasoft Folio Cloud and, of course, your Cloud App.

But aside from monetary aspects, there are also the following benefits to consider:

- Simplicity, ease of use and style: The very essence of Fabasoft's cloud development paradigm is to make it as simple as possible for users to navigate Fabasoft Folio Cloud. To deliver an exceptional online experience to our users is of paramount importance to us. We want them to enjoy every moment they spend using Fabasoft Folio Cloud and your Cloud App. Therefore we will relentlessly continue to improve the simplicity, usability and style of Fabasoft Folio Cloud.
- Reduced time to market and reduced time to value: Fabasoft Folio Cloud is the greatest piece of software Fabasoft has ever made. We're releasing updates at a pace and in a quality never seen before. And you can do the same with your Cloud App, turning your ideas into success stories in virtually no time at all.
- Agility: It's never been easier to respond to the ever changing requirements of your customers. As a Cloud App developer you can instantly enjoy the benefits of the very latest features of Fabasoft Folio Cloud without having to wait for months or years, as you would if relying on a traditional software platform.
- Flexibility and freedom: With Fabasoft Folio Cloud, you're no longer a slave to rigid departmental software policies, tight budgets and other questionable impediments. As a Cloud App developer, you're in total control of an incredibly powerful software platform so you can build the Cloud App that allows your customers, team members and peers to get their jobs done more efficiently, and to make things easier and more enjoyable for all of us.
- Stability and scalability: Over 20 years of experience in both the public and private sector allowed us to build the rock-solid architecture required to support an infrastructure capable of hosting thousands and thousands of users managing and exchanging billions of documents. You can rely on Fabasoft Folio Cloud, 24/7 and 365 days a year, and never have to worry about infrastructure again.

These are bold statements. But we are absolutely sure that with Fabasoft Folio Cloud we give you the ultimate platform for your Cloud App to thrive on. Give it a try! Oh, and we haven't even mentioned the best reason for building a Cloud App yet: It's really good fun!

1.5 What do you need to get started?

Getting started is easy!

The following check list outlines what you need to do to be able to develop your own Cloud Apps for Fabasoft Folio Cloud:

1. Get a free Fabasoft Folio Cloud account:
   The first step is to sign up for a free Fabasoft Folio Cloud account at `https://www.fabasoft.com/register`. For further information about the registration process and a detailed description of the key features of Fabasoft Folio Cloud refer to the chapter “Diving into Fabasoft Folio Cloud” on page 17.
2. Sign up for Cloud App Development:
   Get a subscription to the Fabasoft Folio Cloud App Development package from the Fabasoft Folio Cloud App Store (see `http://developer.foliocloud.com/editions/shop`). Refer to the chapter “Getting a Cloud App Development subscription” on page 25 to learn more about the Fabasoft Folio Cloud App Development package.
3. Get the Cloud App VDE:
   The Virtual Development Environment (VDE) for Cloud Apps is your own sandbox that you can use for developing and testing your Cloud App. To get a subscription for the Cloud App VDE, point your browser to `http://developer.foliocloud.com/editions/shop` where you will find a link to the app in the Fabasoft Folio Cloud App Store. The chapter “Getting a Cloud App VDE subscription” on page 25 explains how to get a subscription for the Cloud App VDE, and in the chapter “Working with the Cloud App VDE” on page 35 you will learn everything you need to know about it.
4. Get the Eclipse IDE:
   Eclipse can be downloaded free of charge from the Eclipse web site [Ecli11]. For further information refer to the chapter “Installing the Eclipse IDE” on page 33.
5. Install the Fabasoft app.ducx plug-in for Eclipse:
   Fabasoft app.ducx is the development environment for implementing Cloud Apps. Add an Eclipse update site and point it to `http://update.appducx.com` to download and install the Fabasoft app.ducx plug-in for Eclipse. To learn how to install the Fabasoft app.ducx plug-in for Eclipse refer to the chapter “Installing the Fabasoft app.ducx plug-in” on page 33.
6. Install Fabasoft app.test Studio primo:
   Fabasoft app.test is the tool for creating and managing automated tests for your Cloud Apps. To learn how to install Fabasoft app.test Studio primo refer to the chapter “Installing Fabasoft app.test Studio primo” on page 138.

That’s it! These six simple steps will get you on your way. And from there, it won’t be long until your first Cloud App goes live in Fabasoft Folio Cloud.

Figure 1: The steps for getting started

By the way, links to all those resources mentioned above can be found in the public team room named “Fabasoft Folio Cloud” (see chapter “Fabasoft Folio Cloud update cycle” on page 26). In the folder “Fabasoft app.ducx for Cloud Development” in this team room you will also find a preconfigured Eclipse IDE with the app.ducx plug-in installed. To access this team room, search for a team room named “Fabasoft Folio Cloud” or use the following URL:
`https://folio.fabasoft.com/folio/mx/COO.6505.100.2.530437`

1.6 Which software do you need on your computer?

This is all you need in terms of locally installed software to start developing Cloud Apps:

- Mozilla Firefox 4.0
  or
  Microsoft Internet Explorer 8.0 or 9.0
- Oracle Java SE Runtime Environment 6 Update 23 (JRE) (see [Orac11a])
- Eclipse 3.6.1 (see [Ecli11])
- Fabasoft app.ducx plug-in (see chapter “Installing the Fabasoft app.ducx plug-in” on page 33)
- Fabasoft app.test Studio primo (see chapter “Installing Fabasoft app.test Studio primo” on page 138)

1.7 What is covered by this book?

This book will give you a solid overview of Cloud App development for Fabasoft Folio Cloud.

After reading this, you will have all the information you need to be able to develop your own Cloud Apps and have them deployed in Fabasoft Folio Cloud.

Nevertheless, while trying to cover all the relevant aspects of Cloud App development, this book is not the compendium of all human knowledge about it.

Therefore, we strongly recommend considering the following reading material:

- For an in-depth discussion of Fabasoft app.ducx and its domain-specific languages for Cloud App development refer to [Faba11a].
- To learn all the details and specifics about Fabasoft app.test refer to [Faba11b]. Furthermore, have a look at [Faba11c] to learn how to create solid tests for your Cloud App.

Also, refer to the list of resources in chapter “Getting help, code samples and support” on page 151 if you run into a problem or need further help.

1.8 General remarks concerning examples

The examples used throughout this book contain code fragments that were specifically created as examples to highlight the use of a particular concept or aspect of Cloud App development with Fabasoft app.ducx or Fabasoft app.test.

Please be aware that not all examples in this book are completely self-contained. In order to save space, certain parts have been omitted in some of the examples. Omissions are usually indicated by a line of dots.

Also, comments in the source code are only included where we want to highlight newly introduced concepts. In repeating code examples, comments are generally applied to the latest additions only.

However, the full source code for all the samples presented in this book, along with many other useful samples and the source code of actual Cloud Apps that are in production in Fabasoft Folio Cloud, are available to you in the public Subversion repository of Fabasoft. For further information refer to the chapter “Retrieving code samples from the public Subversion repository” on page 151.

Diving into Fabasoft Folio Cloud

2 Diving into Fabasoft Folio Cloud

The very first thing you need in order to get started is to get a Fabasoft Folio Cloud account at `https://www.fabasoft.com/register`.

Did we mention already that it's entirely free to get a Fabasoft Folio Cloud primo account?

No hidden costs, no ads, no nag screens.

2.1 Fabasoft Folio Cloud editions

Fabasoft Folio Cloud is based on a freemium model. The Fabasoft Folio Cloud primo edition, encompassing all basic features needed for efficient online collaboration, is entirely free of charge while the premium editions, with advanced features, functionality, more Cloud Apps and more online storage space, come at a charge.

Here is an overview of the different editions of Fabasoft Folio Cloud:

- Fabasoft Folio Cloud primo is the free, basic platform for online collaboration including 1 GB of online storage and a limit of 1,000 objects. Fabasoft Folio Cloud primo includes team rooms for secure online collaboration, management of multimedia contents, Microsoft Office, OpenOffice.org and Apple iWork documents, CalDAV calendar integration, WebDAV and CMIS support and much more. Its intuitive web interface is available in 16 languages, and it supports encrypted connection via HTTPS for security.
- Fabasoft Folio Cloud allegro is our comfort edition with more online storage and mobile search for devices like the Apple iPhone or BlackBerry.
- The Fabasoft Folio Cloud leggero is the professional platform for online business collaboration featuring CRM, two-factor authentication, complete versioning and auditing. Security levels (Protective Markings) and secure team rooms allow for the utmost in online collaboration security.

The Fabasoft Folio Cloud app store is available to all users of Fabasoft Folio Cloud. Also users of the free Fabasoft Folio Cloud primo edition can buy your Cloud Apps from the Fabasoft Folio Cloud app store once your Cloud App is released.

2.2 Registration and account setup

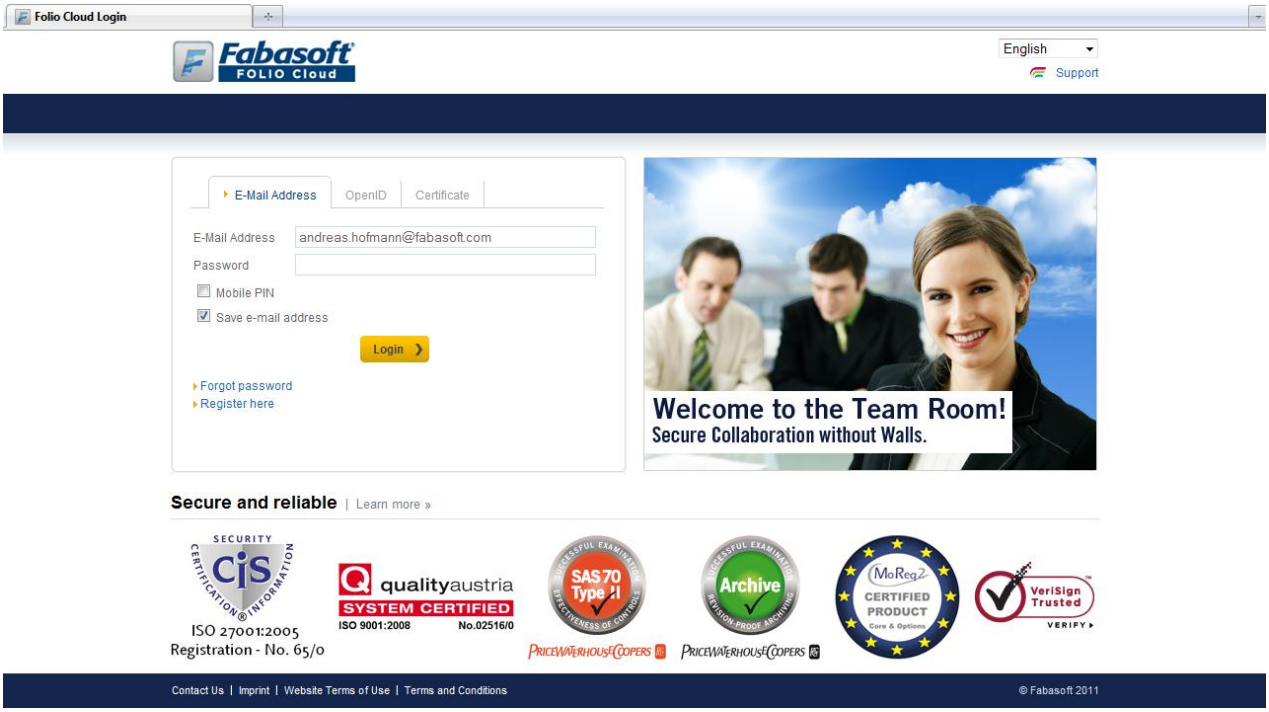


Figure 2: The login screen of Fabasoft Folio Cloud

To start the registration process for Fabasoft Folio Cloud, go to `https://www.fabasoft.com/register`, enter your name, e-mail address and country and click "Next". During the registration process, a verification e-mail is sent to the e-mail address you provided asking you to confirm it and to create a password.

After creating your password you will receive a confirmation e-mail saying that you have successfully completed the registration. The e-mail also contains some useful links to quick tours and other resources.

To log in to Fabasoft Folio Cloud, point your browser to `https://folio.fabasoft.com/folio`, enter your username and password and click "Login" (see Figure 2).

2.2.1 Logging in using an OpenID account

You can also use your OpenID account to log in to Fabasoft Folio Cloud.

OpenID is an open standard for the decentralized authentication of users. An OpenID takes the form of a unique URL managed by an OpenID provider that handles the authentication process.

The following OpenID providers are supported by Fabasoft Folio Cloud: `myopenid.com`, `myid.net`, `flickr.com`, `verisignlabs.com`, and `clavid.com`.

Before you can use your OpenID account to log in, you have to provide your OpenID URL in the account settings dialog of Fabasoft Folio Cloud. To open the account settings dialog, click "Account" in the upper right area of the Fabasoft Folio Cloud portal and click on "Login and Password". Then enter your OpenID URL in the *OpenID* property and click "Change OpenID" (see Figure 3).

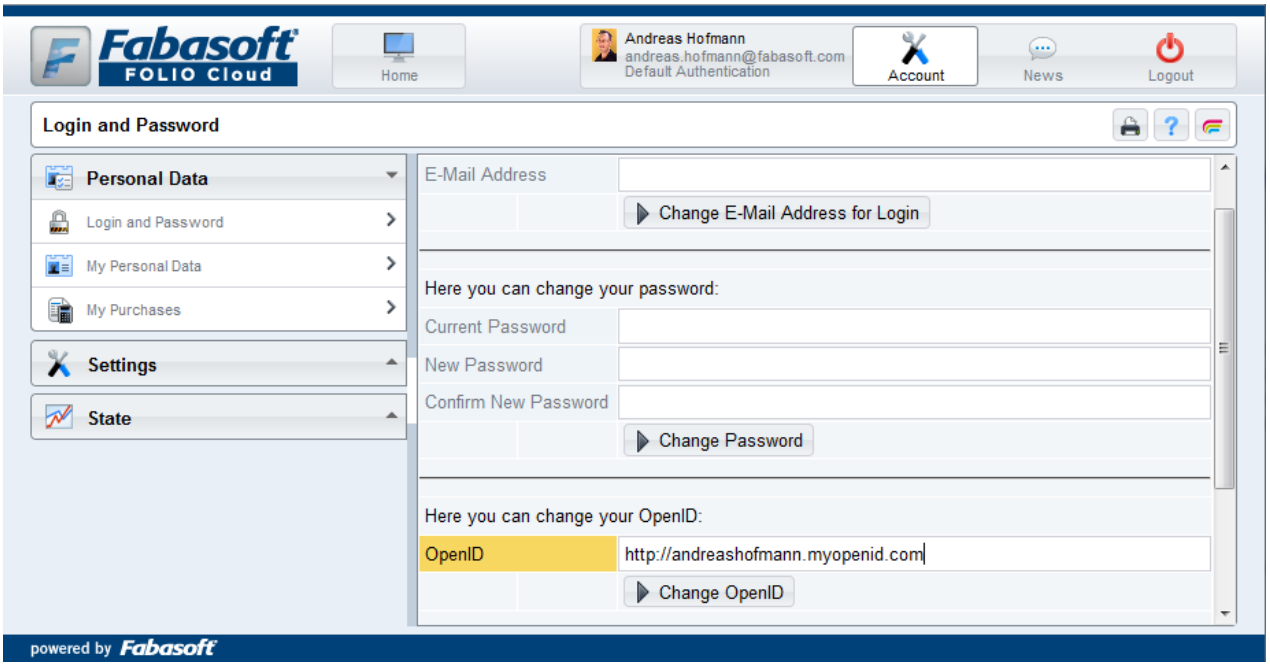


Figure 3: Entering your OpenID account name in the account settings

2.2.2 Logging in using a certificate

Fabasoft Folio Cloud also supports certificate-based authentication for the members of an organization. For more information on certificate-based authentication refer to the Fabasoft Folio Cloud online help.

2.2.3 Logging in using mobile PIN

In addition to one of the basic authentication methods, you can also request mobile PIN authentication for additional security.

With activated mobile PIN authentication, a mobile PIN is sent to your mobile phone via SMS every time you start the login process. In order to continue, you have to provide the one-time passcode sent to your mobile phone.

Mobile PIN authentication is included in Fabasoft Folio Cloud leggero edition and higher (see chapter “Fabasoft Folio Cloud editions” on page 17).

On Apple iPhones Fabasoft Folio Cloud also supports Motoky authentication, a secure mobile authentication method where you don’t need to enter a PIN. You just accept a message sent to your phone via Apple push notification.

For further information on mobile PIN authentication refer to the Fabasoft Folio Cloud online help.

2.3 The Fabasoft Folio Cloud portal

After logging in, you are taken to the Fabasoft Folio Cloud portal, where you are greeted with a welcome screen showing you all the news about your account and your account activity.

The following chapters will give you a brief overview of the main portal pages of the Fabasoft Folio Cloud portal. For further information refer to the Fabasoft Folio Cloud online help and the quick tours available at `http://www.foliocloud.com/quick-tour`.

2.3.1 The ‘Home’ portal page

The “Home” portal page (see Figure 4) is the primary work area. Here you can create new team rooms for online collaboration, folders, documents and other objects.

Figure 4: “Home” portal page of the Fabasoft Folio Cloud portal

2.3.2 The ‘Contacts’ portal page

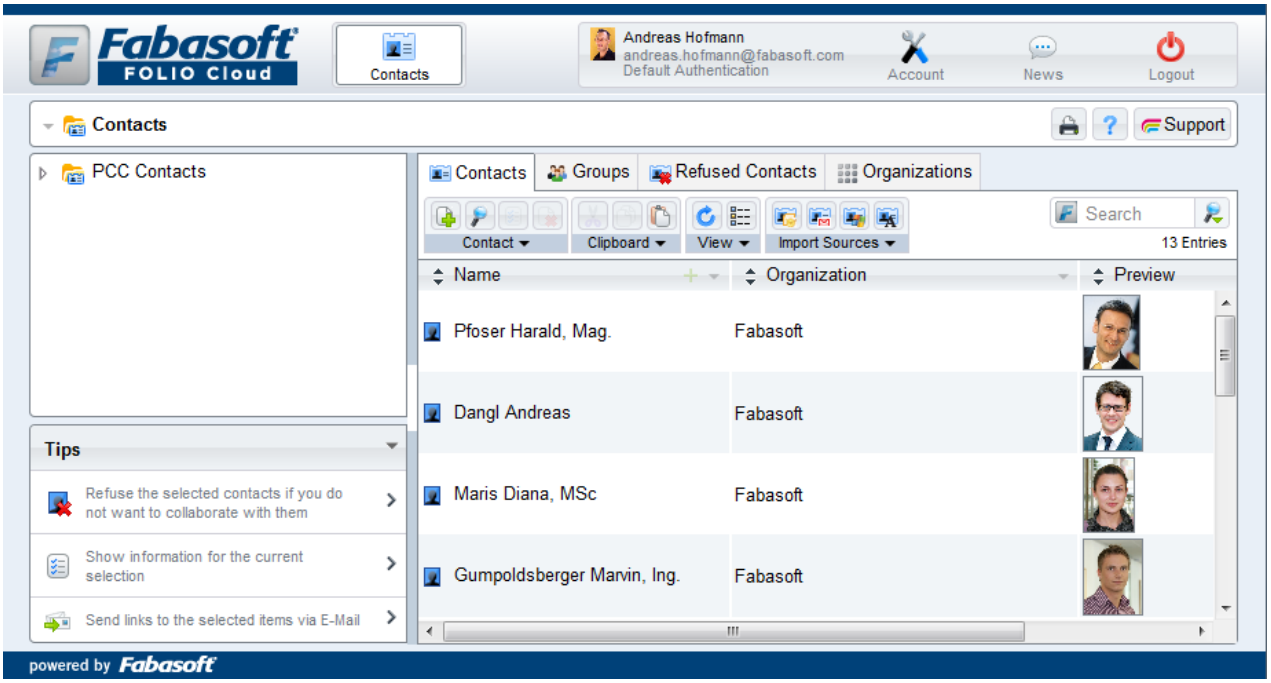


Figure 5: “Contacts” portal page of the Fabasoft Folio Cloud portal

On the “Contacts” portal page (see Figure 5), you can manage your contacts and invite other users to Fabasoft Folio Cloud. You can search for persons and add them into your contact list, or import your existing contacts from Microsoft Outlook, Google Mail, Windows Live Hotmail or other sources. It is also possible to remove a person from your contact list.

2.3.3 The ‘Mindbreeze’ portal page

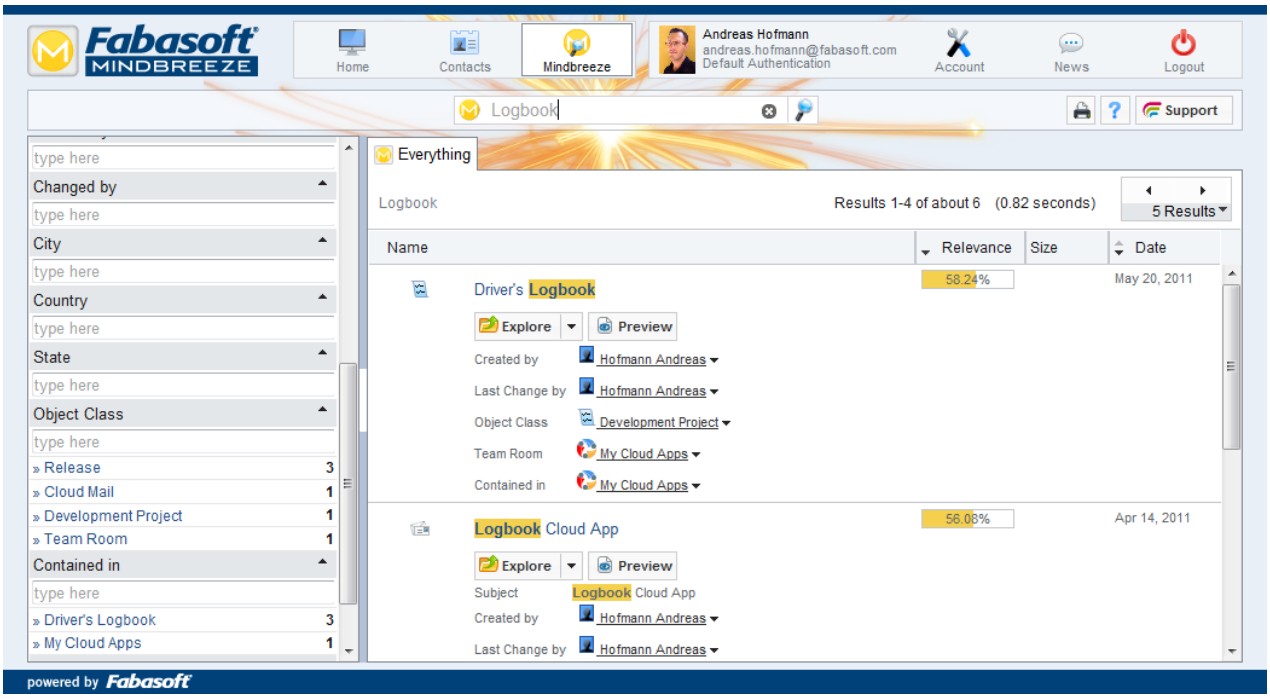


Figure 6: “Mindbreeze” portal page of the Fabasoft Folio Cloud portal

On the “Mindbreeze” portal page (see Figure 6), you can quickly and conveniently search in the full text of all of your documents and objects in Fabasoft Folio Cloud using the integrated Fabasoft Mindbreeze search client.

2.4 Sharing contents with others

You can conveniently share your online contents with others by creating team rooms and inviting other persons to your ad-hoc teams.

All the documents, folders and other objects belonging to a team room are accessible by all members of the team room. As the owner of a team room, you can give others read access, change access or full control. Users who have full control over a team room can also delegate access rights to others on your behalf.

If you want to work together with a person not registered for Fabasoft Folio Cloud yet, you can invite them to join via a team room.

The “Cloud App Team Room” quick tour (available at `http://www.foliocloud.com/quick-tour`) walks you through the steps necessary to create and manage a team room, and shows you how to securely share documents with other persons.

2.5 Join the community!

With your Fabasoft Folio Cloud account, you can also participate in discussions on the Fabasoft Folio Cloud community web site at `http://www.foliocloud.com/community`. Share your knowledge, best practices and helpful tips with others and get answers to your questions.

On the community web site, you can also find a link to a list of frequently asked questions about Fabasoft Folio Cloud along with detailed answers (see `http://www.foliocloud.com/faq`).

In addition to that, the following community resources are available to you:

- In the Fabasoft Folio Cloud Forum (`http://www.foliocloud.com/support/forum`) users of all backgrounds come together to discuss tips, tricks and knowledge about Fabasoft Folio Cloud.
- In the Fabasoft Folio Cloud Wiki (`http://www.foliocloud.com/support/wiki`) you can find a large repository of articles about Fabasoft Folio Cloud and contribute by creating new articles where you share your own Fabasoft Folio Cloud tips and tricks with others.
- Visit the Fabasoft Folio Cloud Blog (`http://blog.foliocloud.com`) and skim through our Cloud experts' posts presenting the latest features, news about upcoming events and webinars as well as other interesting Cloud topics.
- Subscribe to the Fabasoft Folio Cloud Newsletter (`http://www.foliocloud.com/newsletter`) to get all the latest Cloud news delivered to your inbox.
- Follow us on Twitter (`http://twitter.com/FolioCloud`).

2.6 Invite others to join the Fabasoft Folio Cloud!

If you like Fabasoft Folio Cloud, why not tell others about it and make some money by recommending Fabasoft Folio Cloud to your friends and colleagues?

Participate in the Fabasoft Folio Cloud Affiliate Marketing Program and make a couple of bucks for every recommended user who registers and starts working with Fabasoft Folio Cloud!

For detailed information regarding the Fabasoft Folio Cloud Affiliate Marketing Program refer to `http://www.foliocloud.com/affiliatemarketingagreement`.

Starting your Cloud App development project

3 Starting your Cloud App development project

In this chapter, we explain what you need to do in terms of preparation steps before you can begin with the actual development work. Furthermore, we tell you a little bit about the Fabasoft Folio Cloud update cycle and what it takes to get your Cloud App into the Cloud App Store.

3.1 Getting a Cloud App Development subscription

In order to be able to develop Cloud Apps, you need an active subscription to the Fabasoft Folio Cloud App Development package.

To get a subscription to the Fabasoft Folio Cloud App Development package, point your browser to the Fabasoft Folio Cloud Developer site at `http://developer.foliocloud.com/editions/shop` and select "Cloud App Development". Pick a subscription to the Cloud App Development package, add it to your cart and complete the checkout process (see Figure 7).

At the time of writing, Fabasoft is giving away the Cloud App Development package free of charge, but this may change at a later date – so hurry and get your subscription while it's still free!

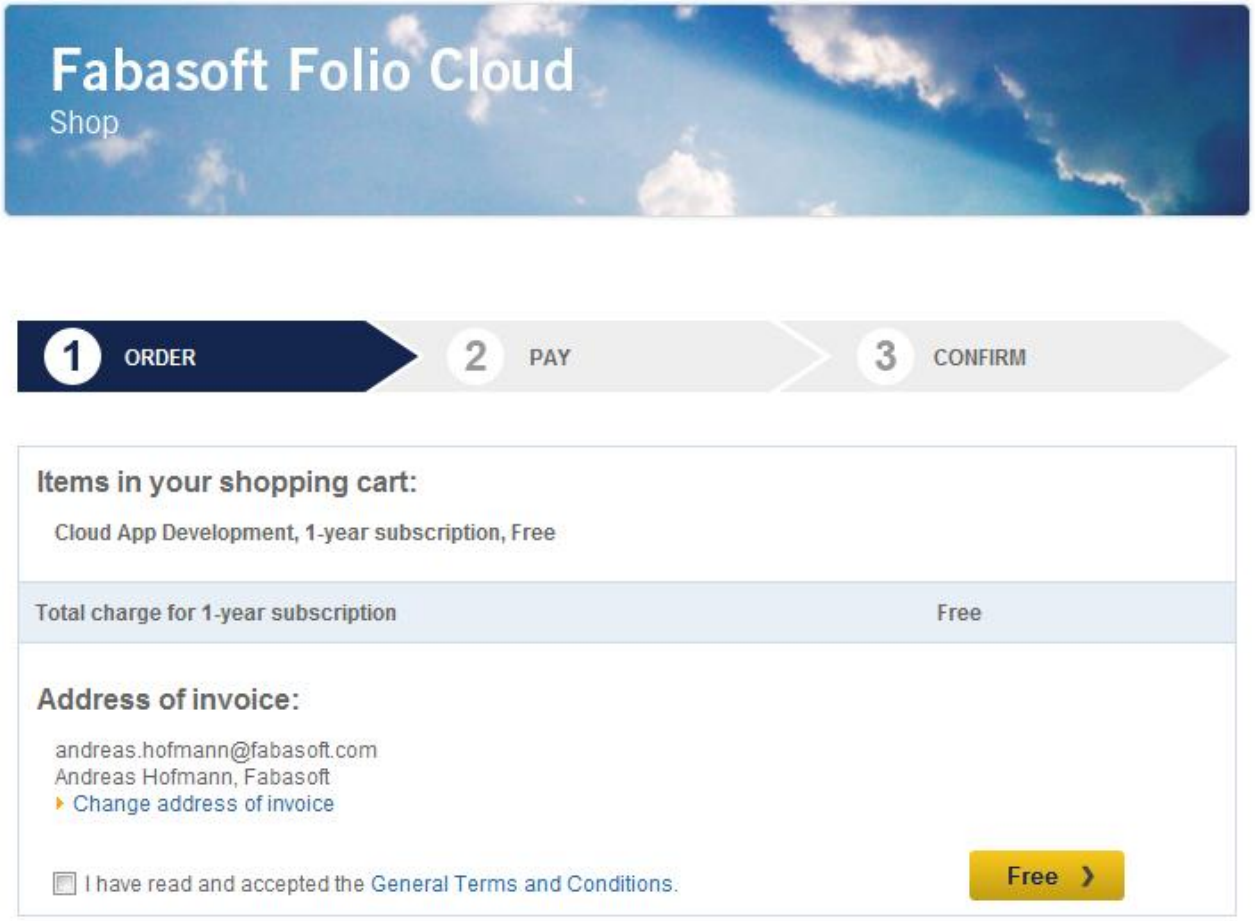


Figure 7: Getting a subscription to the "Cloud App Development" package

3.2 Getting a Cloud App VDE subscription

While the Fabasoft Folio Cloud App Development package discussed in the previous chapter gives you the basic ability to implement Cloud Apps for the Fabasoft Folio Cloud, it does not provide you with a sandbox

so that you can actually try out the stuff you implement in a test environment similar to the actual Fabasoft Folio Cloud.

Indeed, coding is just one piece of the puzzle. Having a sandbox environment to execute and test your Cloud App prototype is another puzzle piece.

The Fabasoft Folio Cloud App VDE (an acronym for “Virtual Development Environment”) is a platform-as-a-service solution for testing and playing around with your Cloud App prototype. It comprises a preconfigured Fabasoft Folio Cloud environment similar to the real production environment where you can deploy, execute and test your Cloud App. Moreover, it includes Fabasoft app.test for creating and running automated tests and Fabasoft app.telemetry for analyzing and pinning down performance bottlenecks.

We believe that you’ll instantly fall in love with the convenience of using the Fabasoft Folio Cloud App VDE as you no longer have to worry about hardware, full hard disks, memory shortages, troublesome software installations and updates. This way, you can put the full focus on your Cloud App while we take care of everything else.

To get a subscription to the Fabasoft Folio Cloud App VDE package, browse to the Fabasoft Folio Cloud Developer site at `http://developer.foliocloud.com/editions/shop` and select “Cloud App VDE”. Pick a subscription to the Cloud App VDE package, add it to your cart and complete the checkout process.

After completing the checkout, you have to create a “Virtual Development Environment” object in Fabasoft Folio Cloud to retrieve the URL of your new Fabasoft Folio Cloud App VDE. For further information refer to the chapter “Working with the Cloud App VDE” on page 35.

3.3 Legal aspects of Cloud App development

As a Cloud App developer you have a lot of freedom, but also some responsibilities.

So there is some legal mumbo jumbo that you have to accept to be able to complete the checkout of the Cloud App Development package subscription.

What it all boils down to is this:

- You must not violate copyright law or any other laws. The source code, template content and other stuff you check in or submit to Fabasoft must be in line with the law.
- Once your Cloud App goes live, you have to actively contribute to supporting it: If something is broken or breaks after an update, you are responsible for fixing it.
- You have to monitor the project inbox of the Scrum project for your Cloud App and process any stories that come in (see chapter “Managing your development project with Scrum” on page 28). Of course, we also encourage you to evaluate and implement user feedback for the improvement of your Cloud App.

Note: The exact details of your rights and responsibilities as a Cloud App developer are detailed in the Fabasoft Folio Cloud developer agreement, available in its latest form at `http://www.foliocloud.com/developeragreement`.

3.4 Fabasoft Folio Cloud update cycle

For Fabasoft Folio Cloud, we’re sticking to a release plan that is carved in stone: Every month, we bring you and the other users an updated version of Fabasoft Folio Cloud so you can always enjoy the latest and greatest that Fabasoft has to offer.

The list of scheduled updates for Fabasoft Folio Cloud is available in the “Fabasoft Folio Cloud” Scrum project that is part of the public team room named “Fabasoft Folio Cloud” (see Figure 8).

To access this team room, search for a team room named “Fabasoft Folio Cloud” or use the following URL: `https://folio.fabasoft.com/folio/mx/COO.6505.100.2.530437`

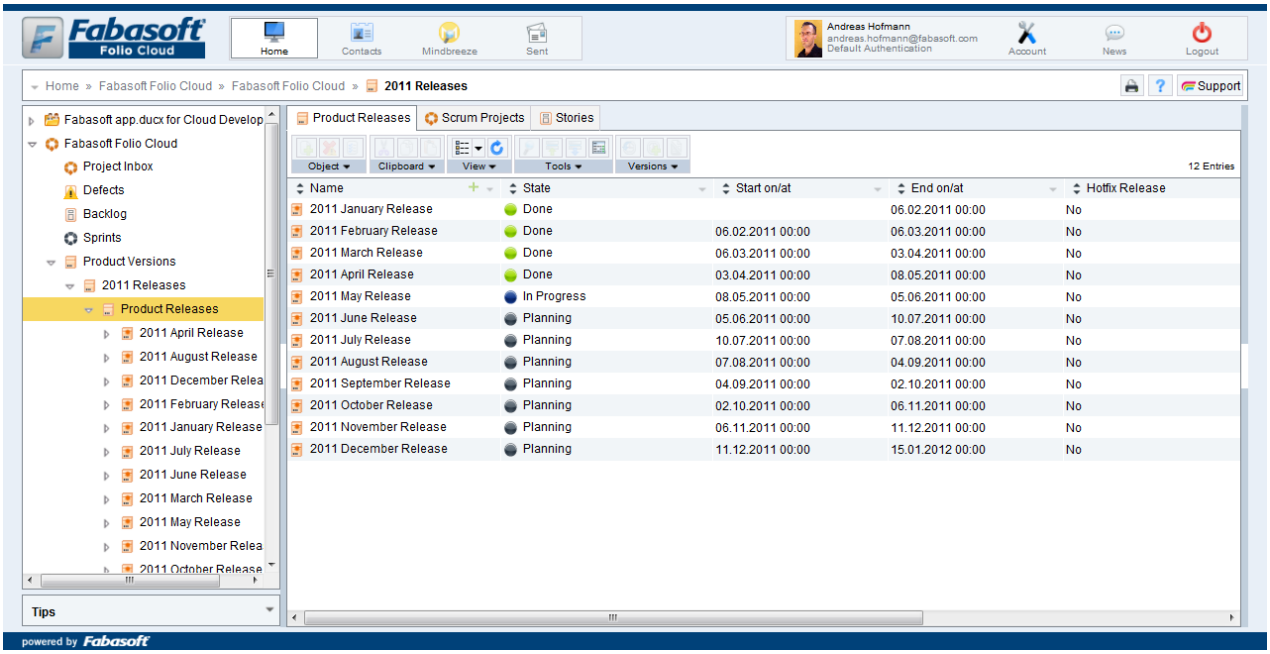


Figure 8: Fabasoft Folio Cloud release plan

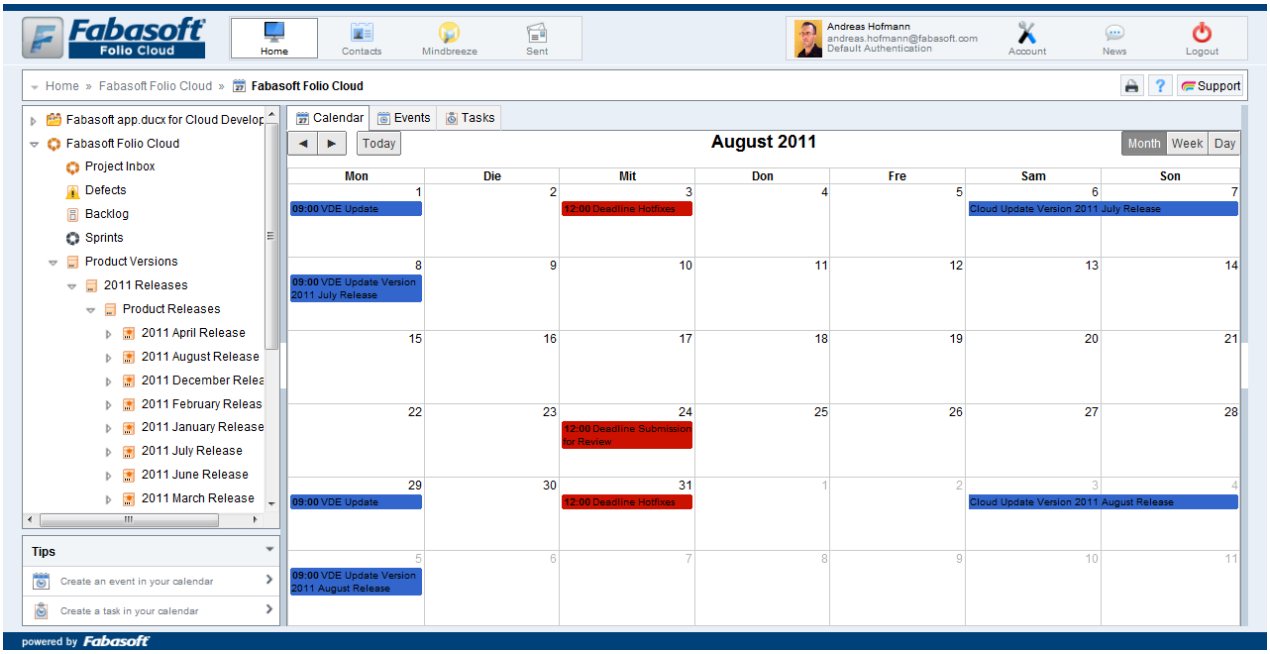


Figure 9: Fabasoft Folio Cloud release calendar

The “Fabasoft Folio Cloud” team room also contains a calendar (depicted in Figure 9) that shows the dates of all important events regarding Fabasoft Folio Cloud App development.

The monthly release cycle of Fabasoft Folio Cloud is not just good news for users, but also for you as a Cloud App developer, as every Fabasoft Folio update is a potential launch window for your Cloud App.

A Cloud App can only go live with one of the scheduled updates of Fabasoft Folio Cloud. If your Cloud App passes the release process outlined in the chapter “Releasing your Cloud App” on page 155, it will be released to the public during the next scheduled update of Fabasoft Folio Cloud.

Note: As a rule of thumb, you must submit your Cloud App about two weeks before the next scheduled update of Fabasoft Folio Cloud for it to be included in that update. Otherwise, it will be included in the update following the next update. For further details refer to [Faba11d].

If you want to release an updated version of a Cloud App that is already live in the Fabasoft Folio Cloud, your update has to go through another release process and again, if successfully completed, will be installed during the next scheduled update of Fabasoft Folio Cloud (see chapter “Releasing an updated version” on page 166).

Despite our joint efforts to produce top quality Cloud Apps, it might happen that a bug is not discovered before your Cloud App is released. Therefore, if a critical defect is discovered in your Cloud App, we may ask you to provide a hotfix that will be applied during a maintenance window outside of the usual update cycle.

3.5 Managing your development project with Scrum

3.5.1 What is Scrum?

According to [ScrA09], “Scrum is an agile framework for completing complex projects. Scrum was originally formalized for software development projects, but works well for any complex, innovative scope of work”.

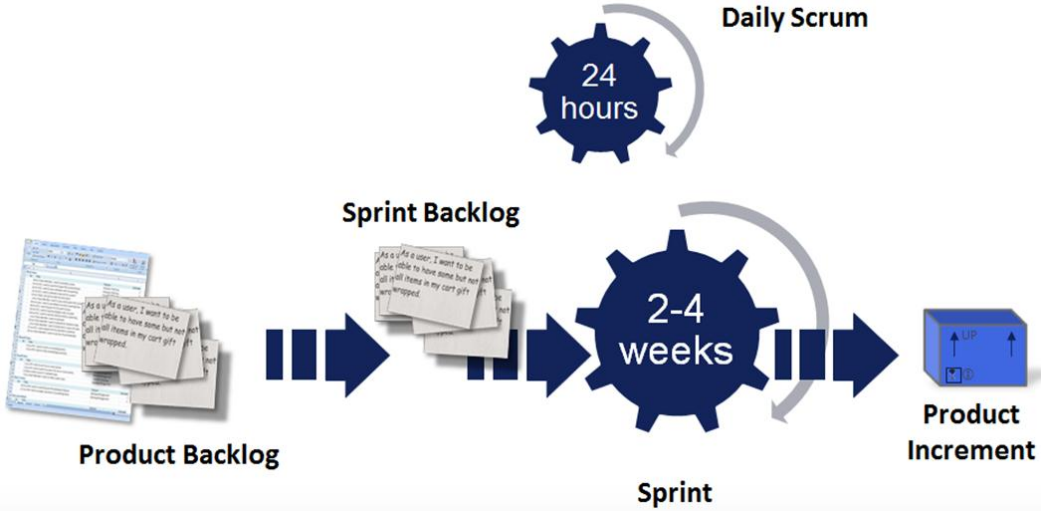


Figure 10: The Scrum methodology

There is an abundance of literature available on Scrum, so we will not elaborate on the methodology itself. [ScrA09] gives you a concise overview of Scrum and is also a good starting point for finding further reading on the methodology.

At Fabasoft, we use the Scrum methodology (outlined in Figure 10) throughout the entire company for developing and managing our software products and services, and we strongly believe that the benefits of this agile state-of-the-art methodology allow you to build better, more reliable software than traditional methodologies.

And then there’s also the advantage of Scrum that we value most of all: It makes you stick to your timetable, which is imperative for Fabasoft Folio Cloud development. There’s an update schedule carved in stone, remember? And in order to meet the deadlines for your Cloud App, so you can capitalize on it as soon as possible, you want a proven, efficient and agile iterative methodology that guides you through the development process.

3.5.2 The “Scrum Projects” Cloud App

Fabasoft Folio Cloud primo edition comes with the full-featured Scrum project management Cloud App that allows you to manage your Cloud App development project.

Even though you are free to pick whatever approach you like for managing your Cloud App development project, we strongly suggest that you follow the Scrum methodology.

We also require you to create a Scrum project for your Cloud App to track any stories associated with it. Whenever a user runs into a problem and sends a support request regarding your Cloud App to Fabasoft Support, a story is created in the Scrum project for your Cloud App and you are required to process it as soon as possible.

For a detailed step-by-step tutorial on how to activate and use the Scrum Cloud App refer to the online help of the Cloud App “Scrum Projects” of Fabasoft Folio Cloud and the quick tour at `http://www.foliocloud.com/apps/scrum`.

3.6 What you need to do to get your Cloud App deployed

Only Fabasoft can deploy Cloud Apps into Fabasoft Folio Cloud.

Before your Cloud App will be deployed into Fabasoft Folio Cloud, you have to submit it to Fabasoft for review.

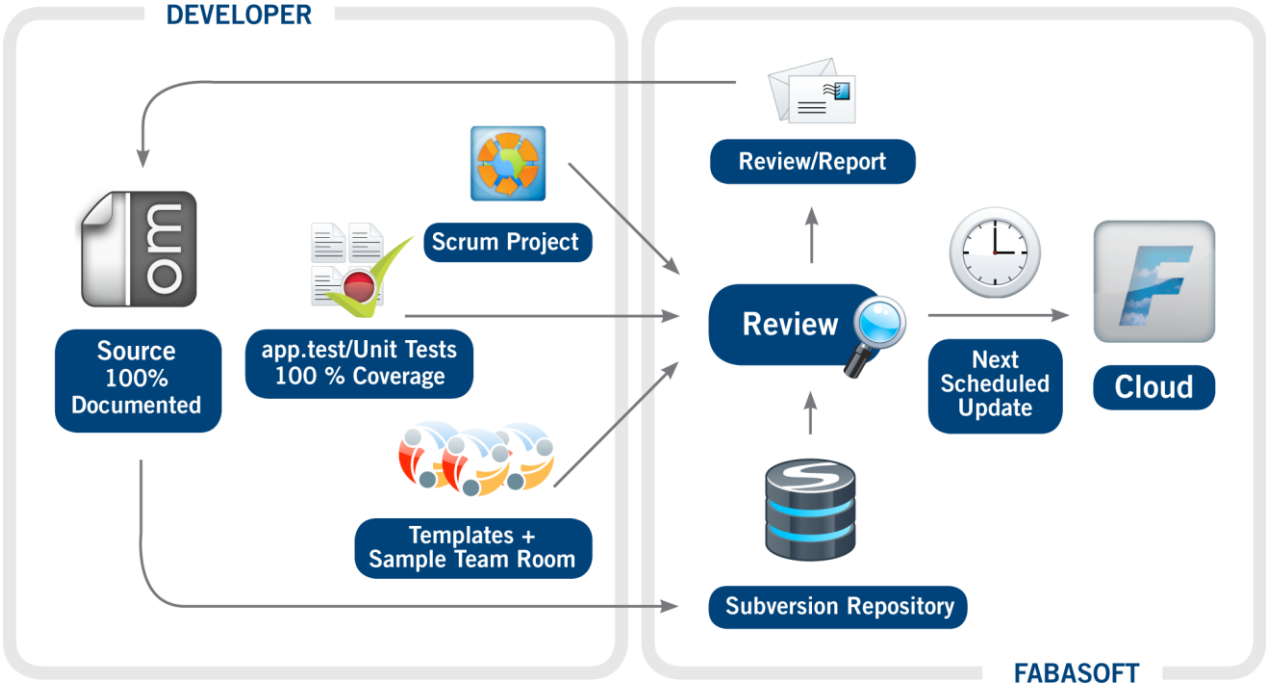


Figure 11: Required deliverables to get your Cloud App deployed

As outlined in Figure 11, the following deliverables are required so that you can submit your Cloud App for review:

- Finished Cloud App: Only a completed, self-contained, full-featured Cloud App may be deployed into Fabasoft Folio Cloud.
- 100 % documentation ratio: Your source code must be fully documented in English language. The target is to reach a documentation ratio of 100 %.
- Zero defects: There must be no open defects in the Scrum backlog for your Cloud App.

- Fabasoft app.test tests and unit tests yielding 100 % code coverage: Your Fabasoft app.test tests and unit tests must cover your entire Cloud App's source code. The target is to reach a coverage ratio of 100 %.

Once you finished your work on your Cloud App and have all the required deliverables in place, you can submit it for review by Fabasoft.

After reviewing your submission, Fabasoft will notify you of the result. If your Cloud App passes the review, it will be included in the next scheduled update of Fabasoft Folio Cloud and the Fabasoft Folio Cloud App Store. If it doesn't pass, an issue report will be created for you so that you can fix the issues pointed out in the report and resubmit your Cloud App for another review.

For further details about the release process refer to the chapter "Releasing your Cloud App" on page 155.

Setting up the development environment

4 Setting up the development environment

Before you can get started with the implementation of your Cloud App, you need to set up your development environment.

More precisely, you have to install Eclipse and the Fabasoft app.ducx plug-in on your computer so you can start coding. Additionally, you need to install Fabasoft app.test Studio primo to be able to create automated tests for your Cloud App.

4.1 Installing the Eclipse IDE

Eclipse is a software development environment featuring an integrated development environment (IDE) and an extensible plug-in system.

Eclipse can be downloaded free of charge from the Eclipse web site `http://www.eclipse.org/downloads` [Ecli11]. Any edition is fine, so you can pick either Eclipse Classic or the Eclipse IDE for Java, PHP or C++ developers.

Note: You can also download a preconfigured Eclipse version from the "Fabasoft Folio Cloud" team room (see chapter "Fabasoft Folio Cloud update cycle" on page 26). For you convenience, the Fabasoft app.ducx plug-in is already pre-installed in this version.

Eclipse requires the Oracle Java Runtime Environment (JRE), which can be obtained from the `http://www.java.com/download` web site [Orac11a].

Eclipse uses so-called features to package plug-ins and allow for full integration with the Eclipse platform. Fabasoft app.ducx provides a feature group that must be installed before it can be used with Eclipse.

4.2 Installing the Fabasoft app.ducx plug-in

Fabasoft app.ducx is packaged as a feature group consisting of multiple Eclipse features and plug-ins. For the sake of simplicity, this book sometimes refers to the entire collection of plug-ins making up Fabasoft app.ducx as "the Fabasoft app.ducx plug-in".

You can install the Fabasoft app.ducx feature by opening the "Help" menu in Eclipse and clicking "Install New Software". Click "Add". In the following dialog, enter `http://update.appducx.com` in the *Location* field and click "OK".

Select the "Fabasoft app.ducx" feature group and click "Next". Then click "Finish" to install Fabasoft app.ducx.

Note: Make sure that your proxy server (if necessary) is configured correctly in Eclipse ("Window" > "Preferences" > "General" > "Network Connections") and that you are connected to the Internet, when installing the app.ducx plug-in (if your Eclipse installation does not meet all prerequisites, missing features are downloaded from the Internet, too).

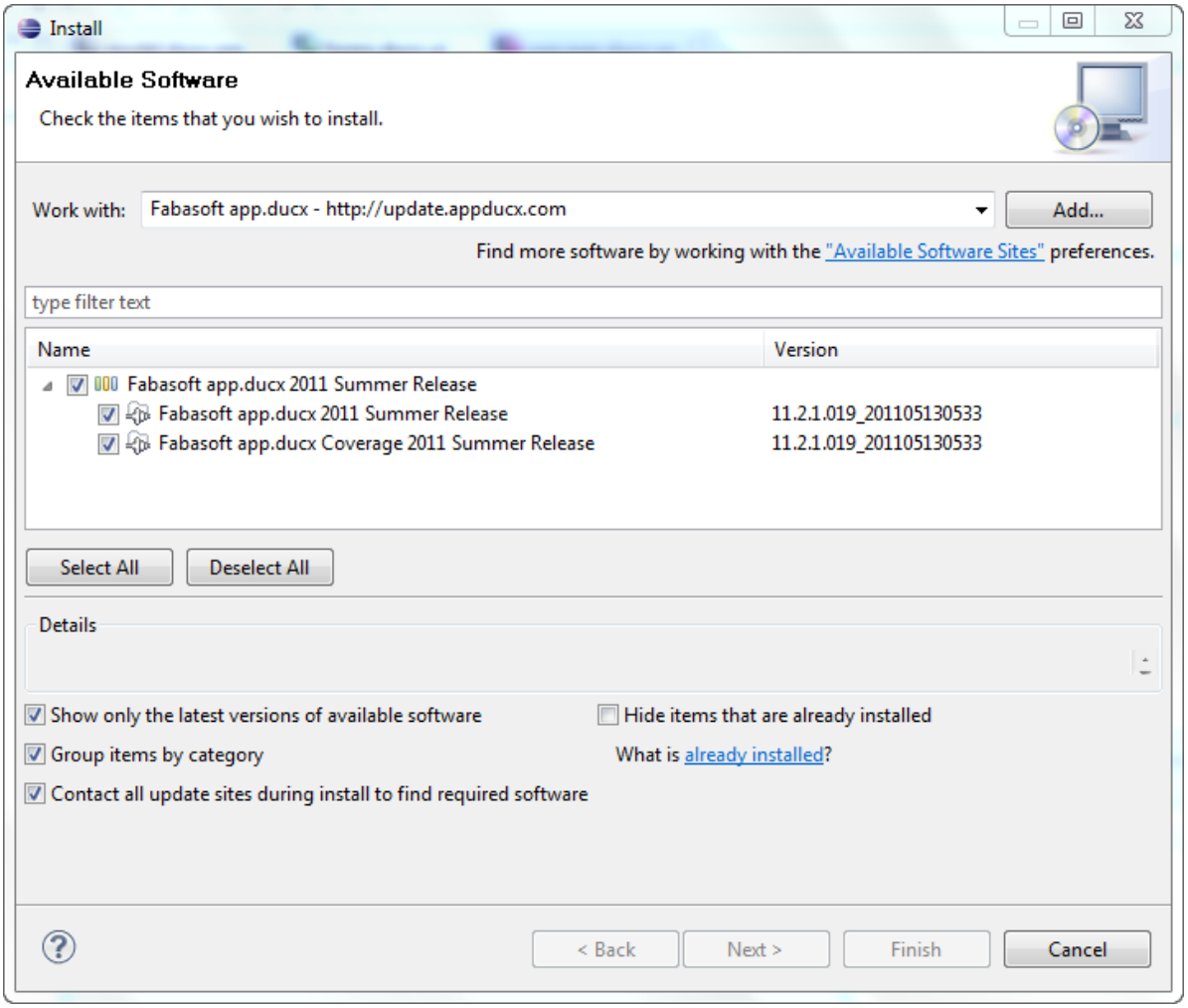


Figure 12: Specifying a new update site in Eclipse

4.2.1 Updating the Fabasoft app.ducx plug-in

The Fabasoft app.ducx plug-in for Eclipse is continuously improved and periodically released by Fabasoft. Every two weeks, the latest version is made available for download from `http://update.appducx.com`.

In order to benefit from all the improvements implemented by Fabasoft, you should always keep your Fabasoft app.ducx plug-in up to date and regularly check for updates.

To update the Fabasoft app.ducx plug-in installed on your local computer, simply select “Check for Updates” from the “Help” menu in Eclipse and follow the steps of the update wizard.

4.2.2 Improving the performance of Eclipse

To improve the performance of your Eclipse environment, edit the `eclipse.ini` file found in the Eclipse installation folder and assign sufficient memory to the Java virtual machine by setting the `-Xms` and `-Xmx` parameters to an adequate value (e.g. “896m” or “1g”).

If you’re using the 32 bit edition of Eclipse, set the `-Xmx` parameter to at least 900 MB of memory, and for the 64 bit edition to at least 1.5 GB of memory.

The rule thumb is “the bigger, the better” when it comes to memory allocation. If you have sufficient free memory available on your machine, consider using the 64 bit edition of Eclipse.

Additionally we recommend using the so called “G1 Garbage Collector” which also has to be activated explicitly in your `eclipse.ini` file.

Example

```
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
-vmargs
-Xms256m
-Xmx1536m
-XX:+UnlockExperimentalVMOptions
-XX:+UseG1GC
```

4.3 Working with the Cloud App VDE

After getting a Fabasoft Folio Cloud App VDE subscription, you have to create a “Virtual Development Environment” object in Fabasoft Folio Cloud. You can create the “Virtual Development Environment” object on your home screen or in a folder of your choice.

When creating the “Virtual Development Environment” object, you are prompted to enter a password, which you have to use for accessing your Cloud App VDE later on. Then click “Next” to save your changes.

In the next step, select “Request Virtual Development Environment” from the context menu of the “Virtual Development Environment” object to initialize your new Cloud App VDE.

Note: You may only have one Cloud App VDE. Attempting to request additional Cloud App VDEs will fail.

To obtain the URL for accessing the self-service portal of your Cloud App VDE (depicted in Figure 13), double-click the “Virtual Development Environment” object and click the URL displayed in the *Self-Service Portal URL* property.

The URL has the following format: `https://folio.fabasoft.com/dev<X>/vm<Y>/`

Note: The placeholders `<X>` and `<Y>` contain the actual IDs assigned to your personal Cloud App VDE.

When connecting to the Cloud App VDE self-service portal, you are prompted for your credentials. Enter the user name `developer` and the password you have provided when creating the “Virtual Development Environment” object.

The self-service portal allows you to carry out the following tasks:

- Restart Services: This will restart the web services of the Cloud Sandbox. A restart of the web services should solve problem situations where the Fabasoft Folio Cloud portal does not respond anymore, e.g. due to an infinite loop in your Cloud App.
- Restart Virtual Machine: This will restart the entire virtual machine (VM). A restart of the VM should only be required in exceptional cases.
- View Trace Output: This allows you to view the trace output of the tracer. To learn more about tracing refer to the chapter “Tracing and debugging” on page 128.
- Fabasoft Folio Cloud Sandbox: This link is a redirect to your Cloud Sandbox. You can directly access the Cloud Sandbox by clicking the URL displayed in the *Cloud Sandbox URL* property of your development project or using the URL `https://folio.fabasoft.com/dev<X>/vm<Y>/folio` (be sure to replace the placeholders `<X>` and `<Y>` with the actual IDs of your Cloud App VDE)
- Fabasoft app.telemetry: This link starts the Fabasoft app.telemetry management interface, which allows you to monitor your VM and identify performance issues caused by your Cloud App. For further information refer to the chapter “Fabasoft app.telemetry” on page 37.

- Set passwords: This allows you to set the password of your `developer` user as well as of all the test users. You can also change the passwords by selecting “Set Password” in the context menu of your “Virtual Development Environment” object.

Figure 13: Fabasoft Folio Cloud App VDE self-service portal

Note: Your Cloud App VDE is automatically wiped and updated before every update of the Fabasoft Folio Cloud. This will allow you to adjust your Cloud App to the latest version of Fabasoft Folio Cloud before the actual update. Refer to the calendar in the “Fabasoft Folio Cloud” team room to find out when the updates of your Cloud App VDE will take place (see chapter “Fabasoft Folio Cloud update cycle” on page 26). You are also notified via e-mail about upcoming updates.

Fabasoft reserves the right to reset your Cloud App VDE at any time. You can also reset it manually. After an update or reset of your Cloud App VDE, all data is lost.

Therefore, we strongly recommend using Fabasoft app.test tests to create the test data structures you need for testing your Cloud App. Do not create test data structures manually as you will have to recreate all of your test data after every update or reset of your Cloud App VDE.

Also note that your Cloud App VDE will not be backed up, and Fabasoft does not guarantee any service levels for it. The maximum size of the Cloud App VDE, including operating system and services is limited to 15 GB. For further details refer to [Faba11d].

4.3.1 Fabasoft Folio Cloud Sandbox

If you click the “Fabasoft Folio Cloud Sandbox” link in the self-service portal, you are redirected to the Cloud Sandbox (see Figure 14), which is a Fabasoft Folio Cloud installation similar to the production environment, and logged in as a “developer” user.

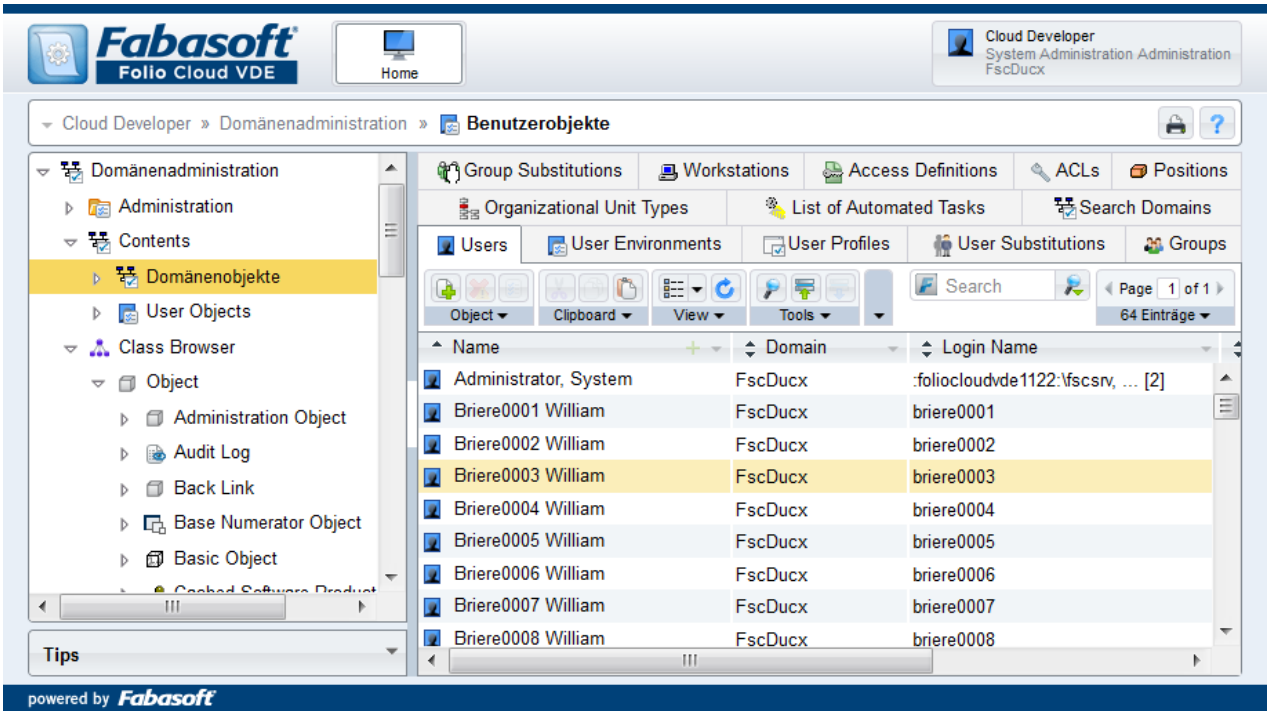


Figure 14: Fabasoft Folio Cloud Sandbox

To log in with a different user account, start a new browser session and use the URL referring directly to the Cloud Sandbox: `https://folio.fabasoft.com/dev<X>/vm<Y>/folio`

There are several preconfigured test users available for accessing the Cloud Sandbox. For the remainder of this book, we will use a user named Wanda Carney with the user account `carney0001`.

Refer to the table in [Faba11c] for a complete list of the available test user accounts. All test user accounts share the same password, which you can define in the self-service portal.

For each test user, there are ten distinct, sequentially numbered user accounts available (e.g. `carney0001`, `carney0002` and so on). The main reason for this is to facilitate the execution of Fabasoft app.test tests in different scopes. For further information on recording and running Fabasoft app.test tests refer to the chapter “Testing your Cloud App” on page 133.

Different test users have subscriptions for different editions of Fabasoft Folio Cloud. If your Cloud App requires a certain edition of Fabasoft Folio Cloud as a prerequisite, pick a test user with the desired edition when conducting manual tests or when creating Fabasoft app.test tests.

Note: The test users do not have access to the Fabasoft Cloud App VDE self-service portal. With a test user account, you can only access the Cloud Sandbox.

4.3.2 Fabasoft app.telemetry

In a nutshell, Fabasoft app.telemetry helps you to identify performance bottlenecks in your Cloud App.

While the full-featured edition of Fabasoft app.telemetry is an integrated solution for service level management, application performance management and helpdesk support that can do much more than that, Fabasoft app.telemetry primo edition, which is included in your Cloud App VDE, is specifically

designed to allow you to track down and resolve potential performance issues in your Cloud App before they can arise in the production environment.

Note: The Fabasoft app.telemetry primo edition only allows you to monitor the requests of the last five minutes.

When you click on the “Fabasoft app.telemetry” link in the self-service portal of your Cloud App VDE, you are redirected to the Fabasoft app.telemetry dashboard depicted in Figure 15:

- On the “Telemetry” portal pane, you can analyze the performance of your Cloud App. Fabasoft app.telemetry automatically records telemetry data for all requests to your Cloud Sandbox as they are being processed by the server. You can then drill down into suspicious or time-consuming requests in order to pinpoint potential performance issues in your Cloud App.
- The “Dashboard” portal pane gives you an overview of the key performance indicators for your VM, including processor time, memory and disk usage as well as the number of requests processed by the Cloud Sandbox and the average request duration.
- The “Status” portal pane provides an overview of the server health checks continuously carried out by Fabasoft app.telemetry to detect any problems with your VM.
- On the “Top X” portal pane, you can analyze the most time-consuming and error prone requests in order to be able to quickly eliminate the biggest performance bottlenecks in your Cloud App.

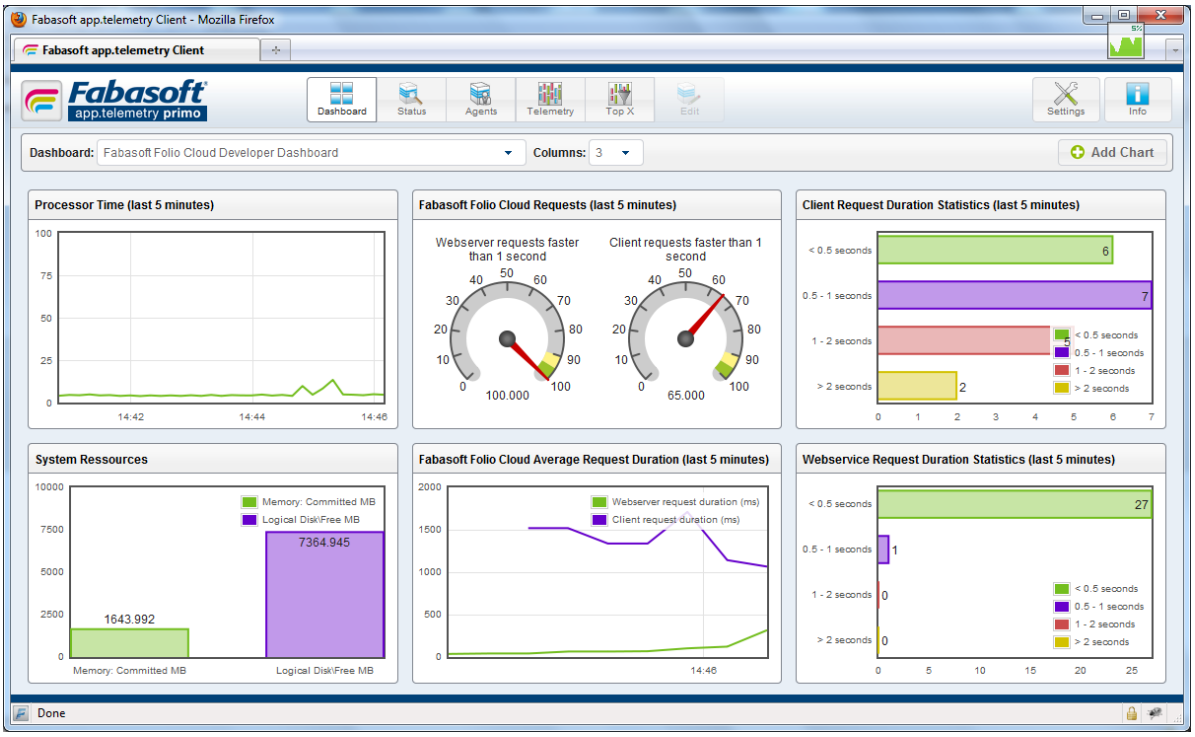


Figure 15: Fabasoft app.telemetry dashboard

For a detailed walkthrough on how to analyze a Fabasoft app.telemetry session refer to the “Fabasoft app.telemetry Getting Started for Developers” guide at `http://www.apptelemetry.com/documentation`.

Creating your Cloud App

5 Creating your Cloud App

In this chapter, we present the sample Cloud App we're going to use for the remainder of this book as well as the steps you need to carry out before you can actually start coding.

In the previous chapters, you've learned all about the preparation of your development environment.

But now it's finally time to get started with your very first Cloud App!

5.1 Introducing your first Cloud App

The sample we've chosen for this book is a driver's logbook to allow users to record the trips they make with their vehicles.

Simply put, we're going to build a Cloud App for recording and tracking vehicle mileage which can be used to get a deduction on your income tax return or to get reimbursement from your company.

Figure 16 shows a simplified UML class diagram of this sample.

The following requirements must be met:

- The object model of this Cloud App consists of two classes, logbook and trip log.
- A logbook contains trip logs, which are collections of recorded trips.
- For each trip, the information listed in Table 1 is recorded.
- Users can create and manage as many logbooks as they wish.
- When creating a logbook, a trip log is automatically created within the logbook.
- Legislation requires that recorded trips must not be changed anymore, but users can cancel the last recorded trip in the trip log.

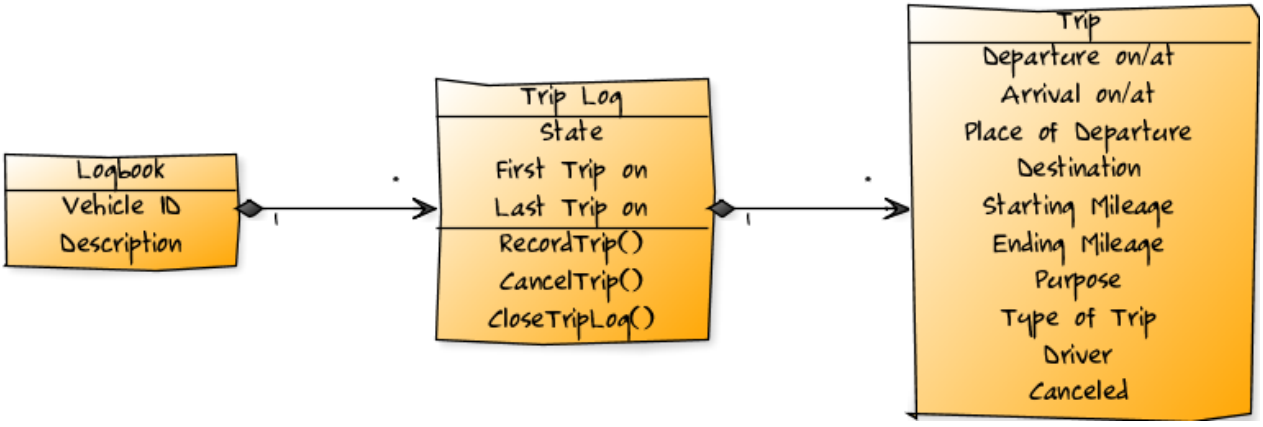


Figure 16: Class diagram of the logbook sample

Property Name	Data Type	Reference	Remarks
Departure on/at	`datetime`	`trpdepartureat`	Date and time when the trip is started
Arrival on/at	`datetime`	`trparrivalat`	Date and time of arrival
Place of Departure	`string`	`trpdepartureplace`	The name of the place where the trip is started
Destination	`string`	`trpdestinationplace`	The name of the destination

Starting Mileage	`float`	`trpstartmileage`	The vehicle's odometer reading before starting the trip
Ending Mileage	`float`	`trpendmileage`	The vehicle's odometer reading upon arrival
Purpose	`string`	`trppurpose`	A brief description of the purpose of the trip
Type of Trip	`object`	`trptype`	One of the following options describing the type of the trip: "Private", "Business" or "Commute"
Driver	`string`	`trpdrivername`	The name of the driver
Canceled	`boolean`	`trpcanceled`	A flag indicating whether the trip was recorded in error and, therefore, canceled

Table 1: Data structure for recorded trips

5.2 Creating the development project

The first steps in a Cloud App development project are carried out from within your Fabasoft Folio Cloud portal. Therefore, browse to `https://folio.fabasoft.com/folio` and login to Fabasoft Folio Cloud.

Then carry out the following steps:

1. Create a new team room by selecting "Create Team Room" from the "Object" menu or by clicking the "Create Team Room" button.
2. Enter a name for your new team room, e.g. "My Cloud Apps", and select "App Development" in the *Type* property, then click "Next" to finish creating your new team room.
3. Open the "My Cloud Apps" team room by double-clicking it and select "New" from the "Object" menu or click the "Create a new object" button.
4. Select "Development Project" from the list of createable objects and click "Next". In the next dialog, enter a name for the new "Development Project", e.g. "Driver's Logbook", and click "Next".
5. Edit the properties of the new development project object (see Figure 17) and enter "FSCLOGBOOK" in the *Reference* property as well as a copyright text and a description.

Every Cloud App needs a unique reference, which is roughly similar to a namespace. Therefore, the reference you provide in the *Reference* property must meet certain naming conventions (e.g. it should be composed of upper case characters only, must not contain special characters and must begin with an alpha-numeric character in the range from "A" to "Z"), and must be unique within Fabasoft Folio Cloud.

For further information on reference naming conventions refer to [Faba11a].

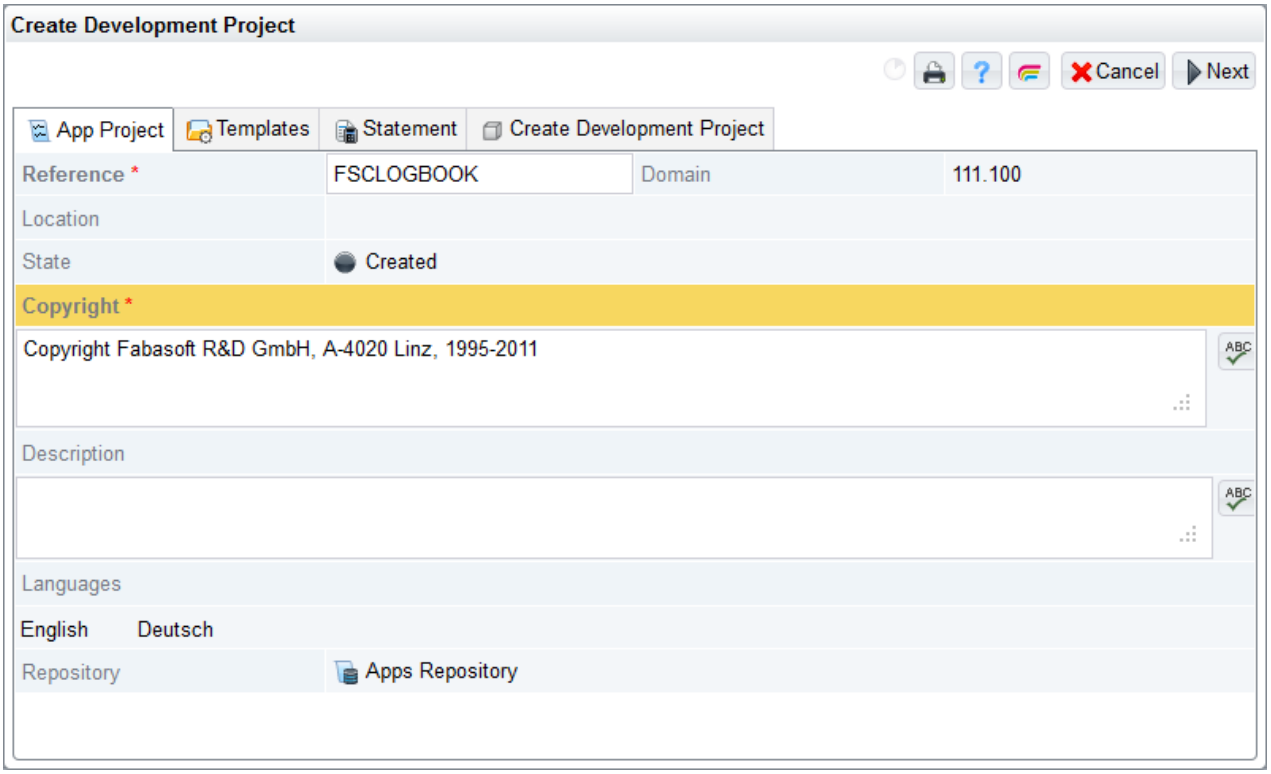


Figure 17: Properties of a “Development Project”

On the “Statement” page, you have to enter all the data required for billing so you can actually get your money when people buy Cloud App subscriptions or Activity Points for your Cloud App. Refer to the chapter “Reaping the profits” on page 159 for all the information about making money with your Cloud App.

In the next step, select “Next” to save your changes and then “Publish” from the context menu of the development project to create a skeleton project for your Cloud App in the Subversion repository of Fabasoft Folio Cloud.

Before the skeleton project is generated, the reference you’ve provided for your Cloud App is checked for compliance with the naming conventions described before. Also, if there is already an existing software component with the same reference you will be asked to choose a different reference. Of course, the reference (as well as the rest of your code) also must not violate copyright law or be offensive in another way.

After successfully publishing your development project, a URL to your project in the Subversion repository is displayed in the *Location* property of the development project.

This URL has the following format:

```
https://folio.fabasoft.com/svn/apps/<unique ID>/trunk/<Cloud App reference>
```

In preparation for the next step, copy the URL that is displayed in the *Location* property of your development project into the clipboard.

For the remainder of this book, we will use the following example URL to access the Subversion repository:

```
https://folio.fabasoft.com/svn/apps/FCB798DAF91D3911AB30860F534FADBA/trunk/FSCLOGBOOK
```

5.3 Creating a release

For every release of your Cloud App, you have to create a release object in the *Releases* list of your development project (i.e. when you start developing and before starting to work on subsequent updates).

To create a release, double-click your development project to open it, select the *Releases* list and click “Create Object”. In the dialog box that is opened, provide the version number for your release by entering the respective values in the properties *Major*, *Minor* and *Build* (e.g. “1.0.0”).

After creating the release, select “Start Release” from its context menu (as shown in Figure 18). This creates an internal marker in the Subversion repository.

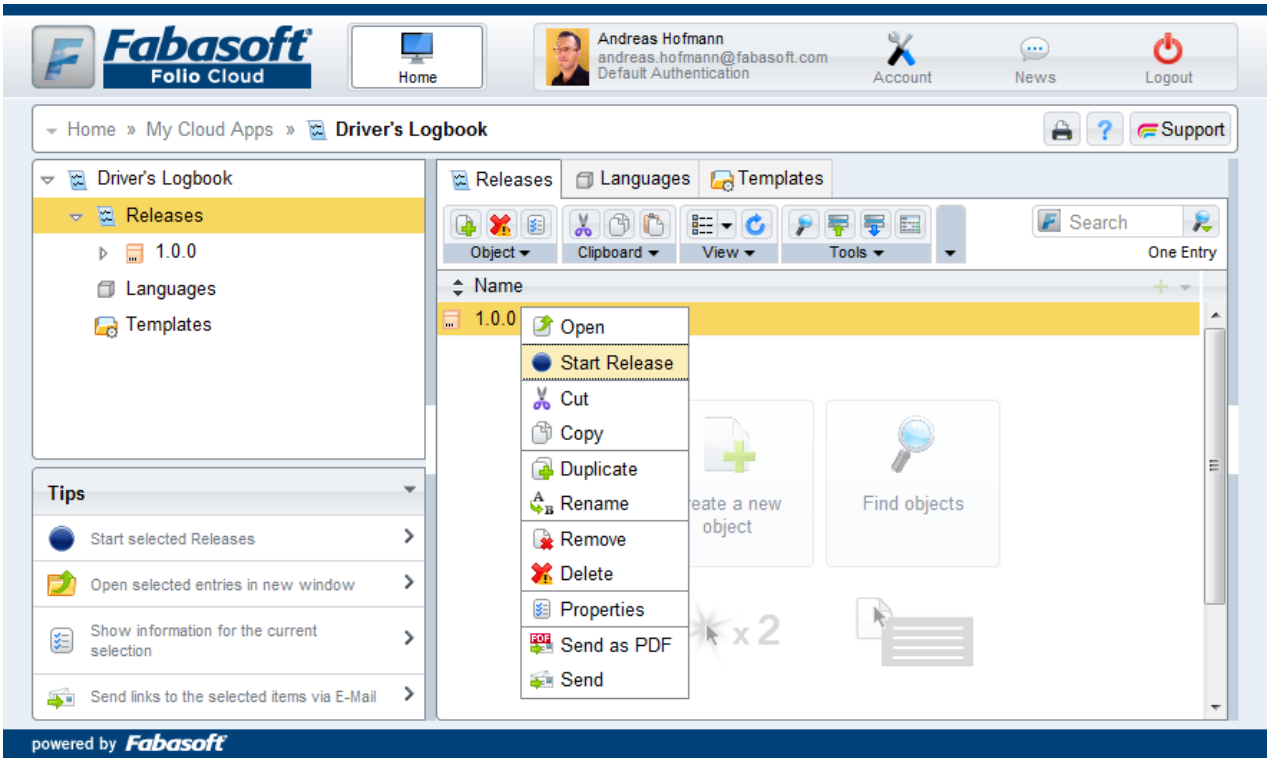


Figure 18: Starting a release

5.4 Importing the Fabasoft app.ducx project in Eclipse

Now it’s time to open Eclipse and import the skeleton project that was created for you.

However, before you import the project you should define the default Fabasoft Folio Cloud web service and the default range service you want to use when developing.

5.4.1 Defining the default web service

Select “Preferences” from the “Window” menu and select the “Fabasoft app.ducx” tree node. Click the “Edit” button next to “Default web service” and enter the URL of your Cloud Sandbox in the *URL of the Fabasoft Folio Web Service* field in the dialog box that pops up.

For the remainder of this book, we will use the example URL `https://folio.fabasoft.com/dev2/vm23/folio`, but you have to enter your own personal Cloud Sandbox URL that was assigned to you.

Select “Basic” as authentication method and enter “developer” in the *User name* field along with the password you assigned to the Cloud App VDE users (see chapter “Working with the Cloud App VDE” on page 35).

Then click “OK” to close the dialog boxes.

5.4.2 Defining the default range service

After defining the default web service for your Cloud Sandbox, you have to define the default range service, which will automatically assign address ranges to you.

Every single object you define in your code is assigned a unique address. These addresses are managed automatically by the Fabasoft app.ducx compiler, but in able to do so it needs to retrieve so-called address ranges from a range service that assigns ranges of free addresses to you.

Click the “Edit” button next to the *Default range service* field and enter the URL `https://folio.fabasoft.com/folio`. Select “Basic” as authentication method and provide your Fabasoft Folio Cloud credentials in the *User name* and *Password* fields. Then click “OK” to close the dialog box.

Note: Pay attention when providing the user credentials! When defining the default web service (as described in the previous step), you have to use the “developer” user of your Cloud App VDE, whereas for the default range service you need to provide your Fabasoft Folio Cloud user credentials.

In the “Preferences” dialog box, select “Automatically request new range”. Finally, click “OK” to save your settings (see Figure 19).

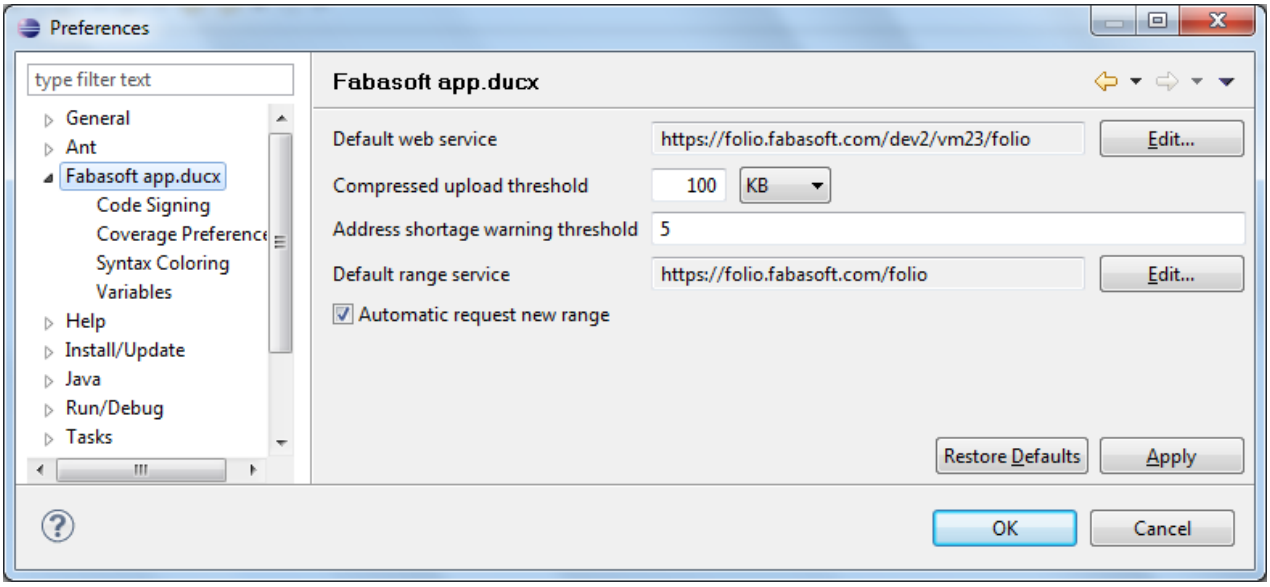


Figure 19: Fabasoft app.ducx preferences

5.4.3 Importing the Fabasoft app.ducx project from Subversion

You can import the app.ducx project directly from Subversion if you have installed “Subversive” – the subversion plug-in for Eclipse (see `http://www.eclipse.org/subversive`).

In Eclipse, select “Import” from the “File” menu. In the “Import” dialog box that pops up, expand the “SVN” branch, select “Project from SVN” and click “Next” (see Figure 20).

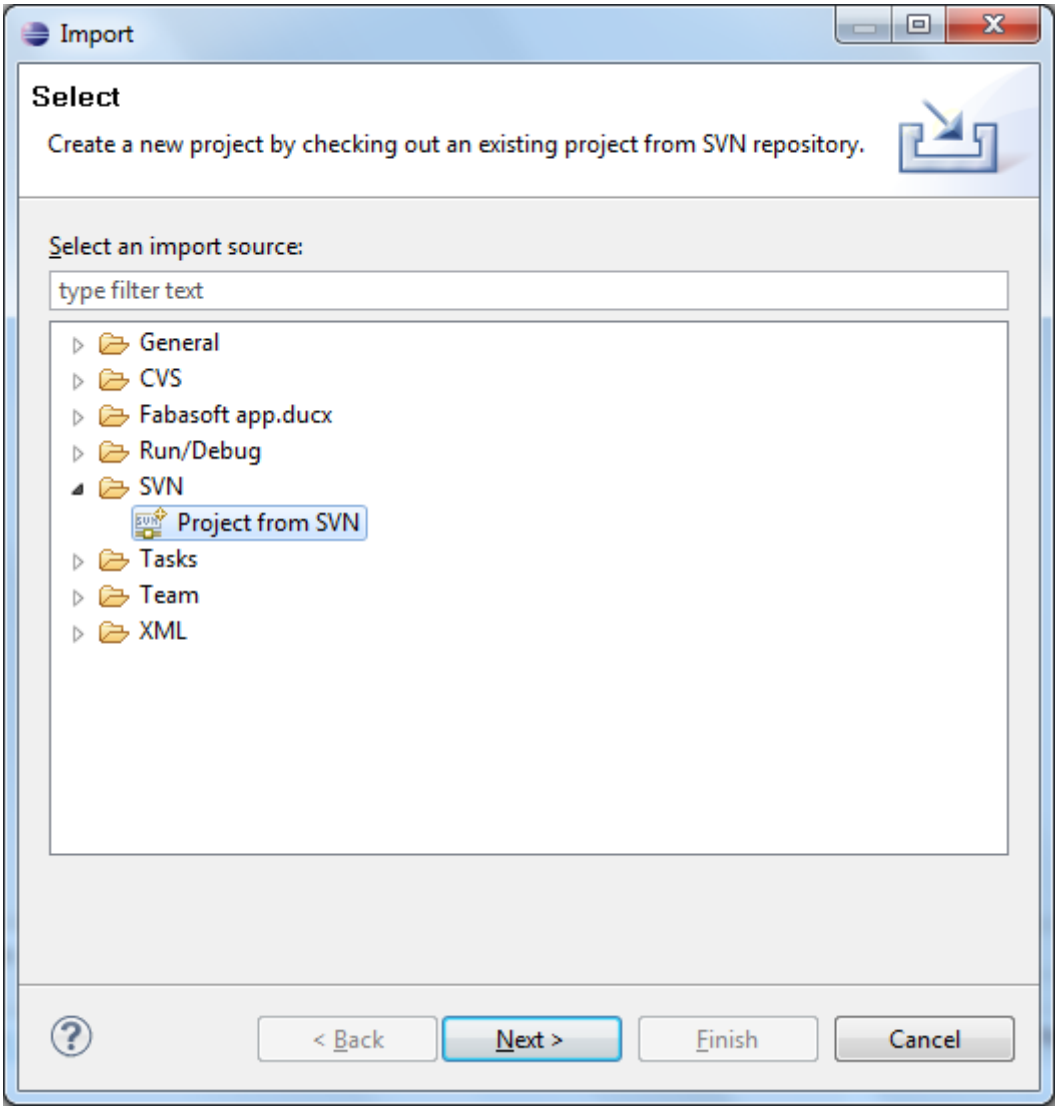


Figure 20: Selecting the Subversion project import wizard

In the “Checkout from SVN” dialog box depicted in Figure 21, paste the URL from the clipboard that is pointing to your Cloud App project in the Subversion repository. Then enter your Fabasoft Folio Cloud credentials in the *User* and *Password* fields of the *Authentication* box and click “Browse” to log in to the Subversion repository.

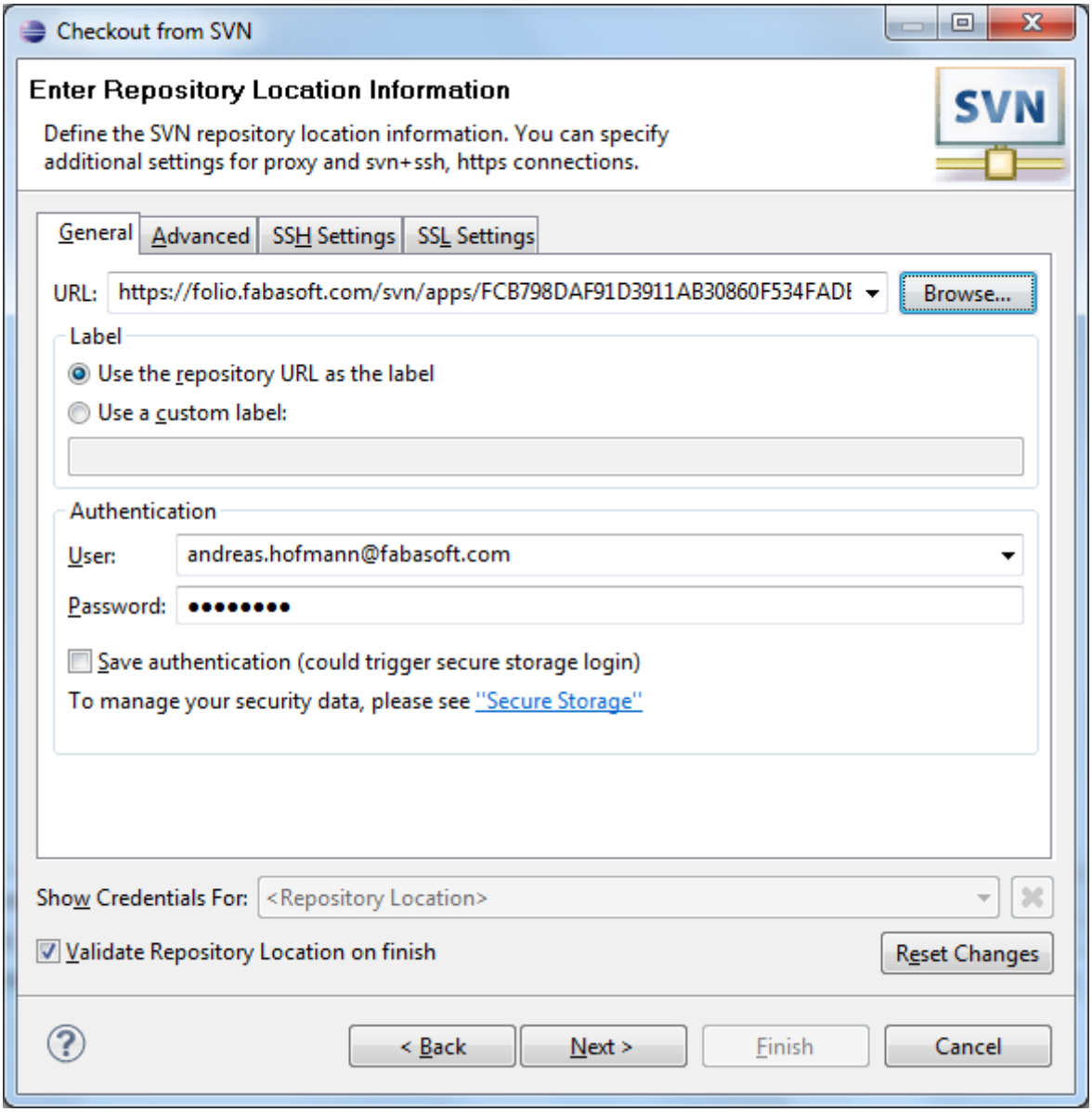


Figure 21: Entering the Subversion repository location information

In the “Select Resource” dialog box depicted in Figure 22, expand the “trunk” branch and the “FSCLOGBOOK” branch underneath it, and select the “dev” branch. Then click “OK” to return to the “Checkout from SVN” dialog box and click “Finish” to proceed with the import of your project from Subversion.

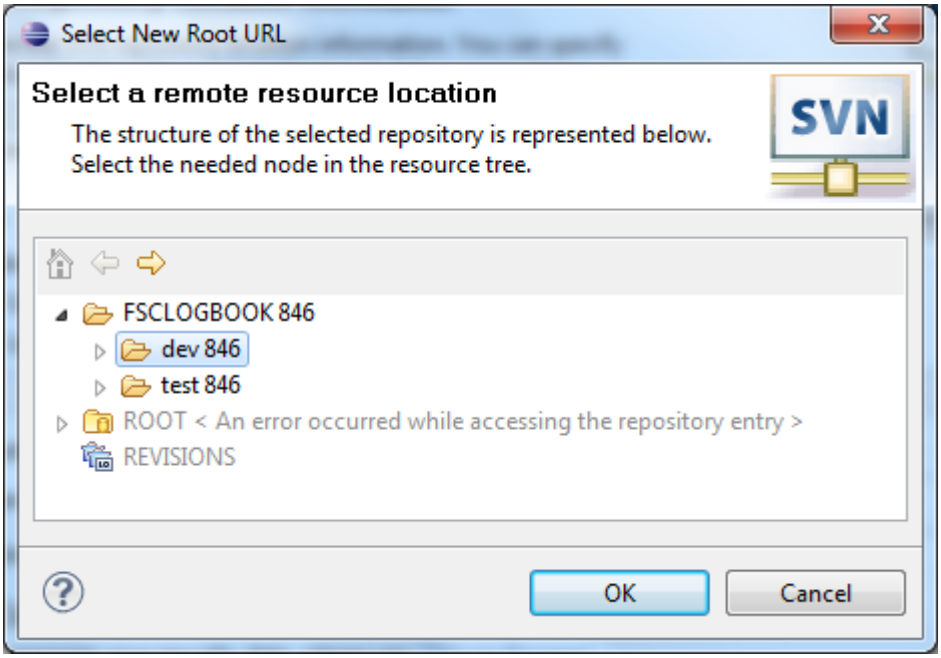


Figure 22: Selecting the “dev” branch

In the “Check Out As” dialog box that is opened next (see Figure 23), confirm the suggested “Check out as project with the name specified” and click “Finish”.

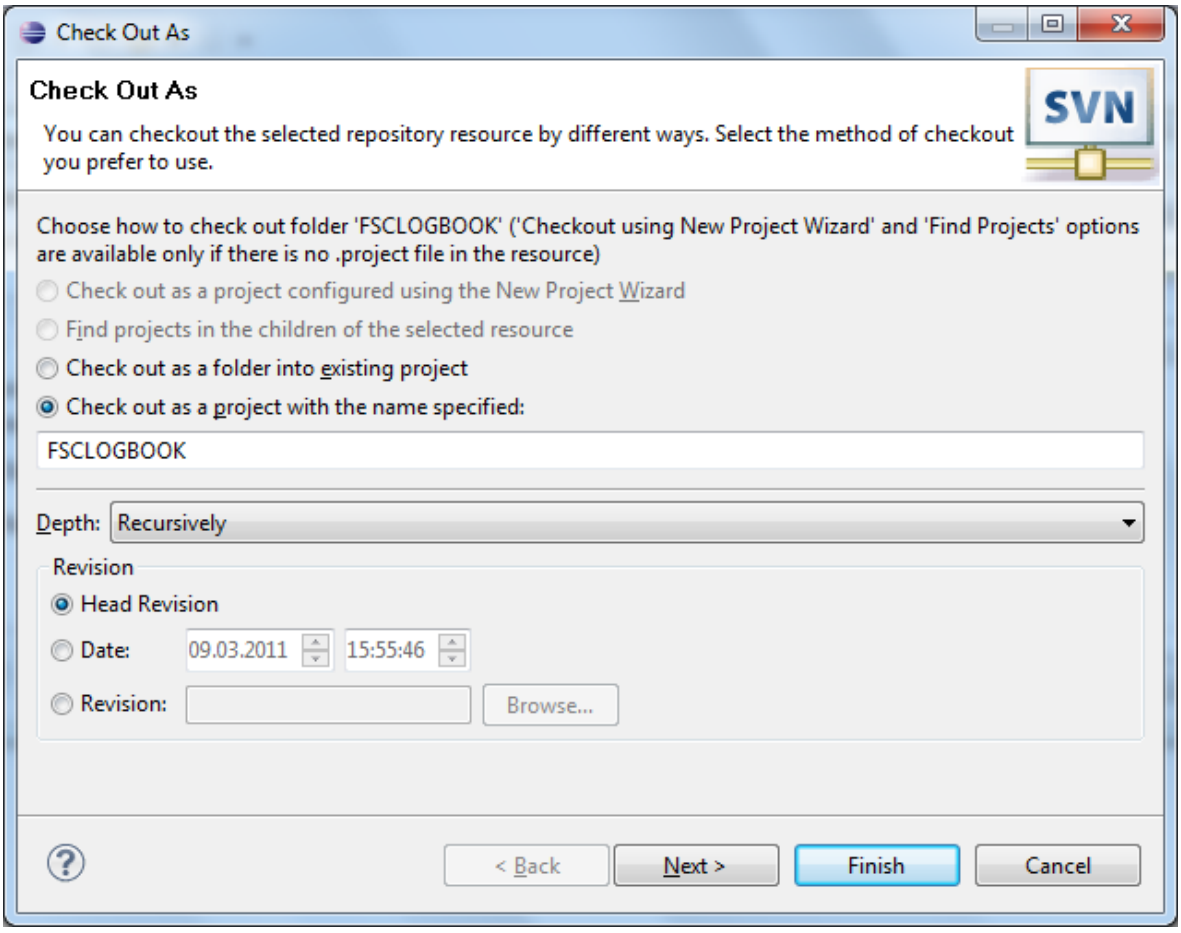


Figure 23: Confirming the project name

5.4.4 Selecting the address range

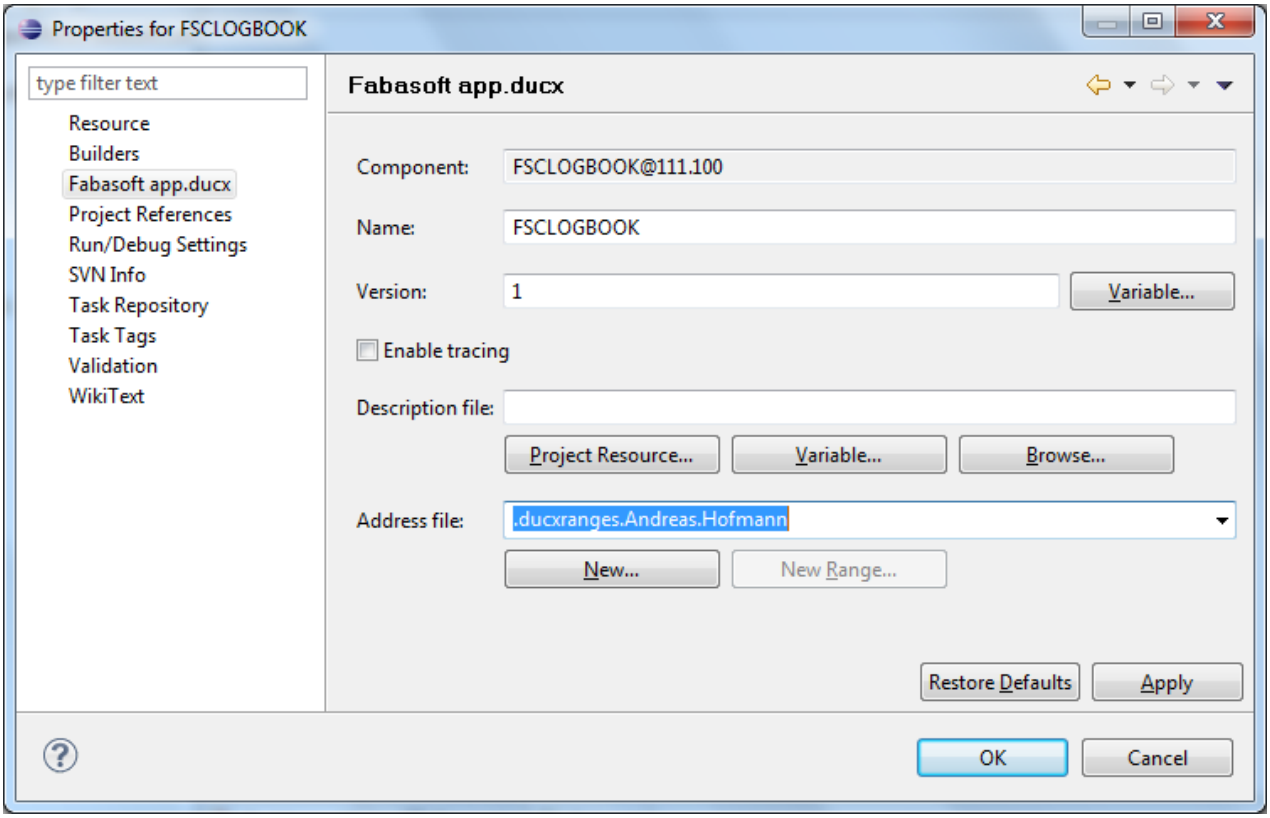


Figure 24: Selecting the address range file

After you have successfully imported your Cloud App project, select “Properties” from the context menu of Project Explorer and select the “Fabasoft app.ducx” tree node in the “Preferences” dialog box (see Figure 24). In the *Address file* field, select the address range file that has been generated for you.

The name of the address range file starts with the `.ducxranges` prefix, e.g. `.ducxranges.Andreas.Hofmann`.

Click “OK” to save your changes.

Note: If multiple developers need to work on the same project at the same time you have to create a separate address range file for each developer. To create a new address range file, click “New” and enter a filename. Before you start working on your project, make sure that your personal address range file is selected in the *Address file* field.

For further information on how to manage address ranges refer to [Faba11a].

5.5 Accessing and managing the source code in Subversion

You can either use the Subversion command line tools to access your source code in the Subversion repository of Fabasoft Folio Cloud or use a graphical Subversion client such as TortoiseSVN for Microsoft Windows (depicted in Figure 25).

To connect to the Subversion repository, use the URL provided in the *Location* property of your development project (see chapter “Creating the development project” on page 42) and your Fabasoft Folio Cloud credentials.

Refer to [ApSF11] for detailed information on how to use Subversion to access and manage your source code in the Subversion repository of Fabasoft Folio Cloud.

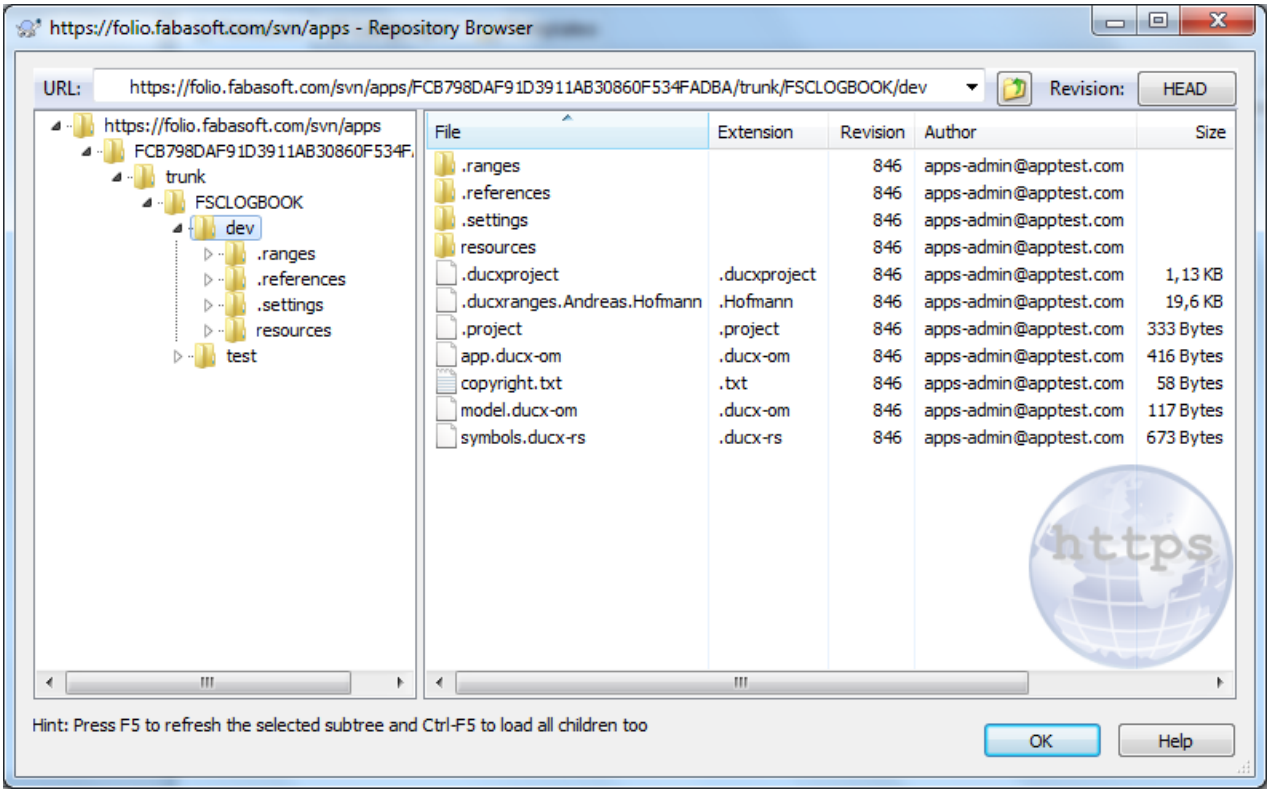


Figure 25: Using the Repository Browser of TortoiseSVN to access your source code

Implementing your Cloud App

6 Implementing your Cloud App

Finally you've arrived at the chapter where we're about to plunge knee deep into coding. So put your seat belt on, pull out the "An Introduction to Fabasoft app.ducx" white paper [Faba11a] and make sure you're in the "Fabasoft app.ducx" perspective in Eclipse (if not, select "Window" > "Open Perspective" > "Other" > "Fabasoft app.ducx").

If you carefully followed all the steps we described in the previous chapters you should now have a skeleton project for your Cloud App in your Eclipse workspace and your screen should somehow look similar to what can be seen in Figure 26.

And now, let the coding begin!

But wait a second… which programming language are we going to use to build our Cloud App?

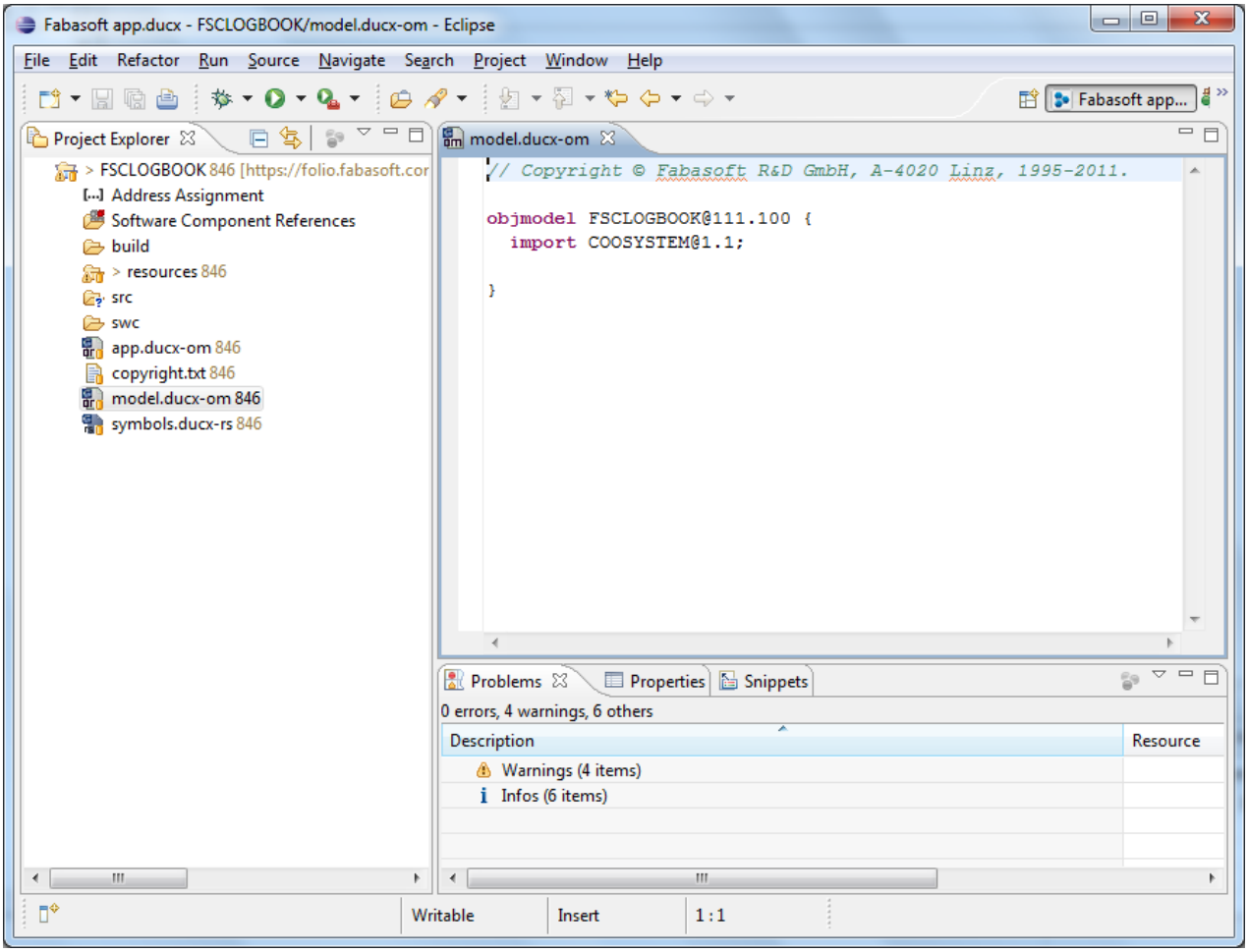


Figure 26: Your Cloud App project in Eclipse

6.1 Introducing the domain-specific languages of Fabasoft app.ducx

Fabasoft app.ducx is based on a set of different modeling languages referred to as domain-specific languages (DSLs), where each DSL was designed for addressing a certain aspect of Cloud App development:

- The purpose of the app.ducx object model language is to define the persistent object model for your Cloud App, such as object classes and properties.
- The app.ducx resource language allows you to define resources such as string objects, error messages and symbols. Using the app.ducx resource language, you can create culture- and language-

independent solutions as it allows you to avoid hard-coded, culture- and language-specific literals in your solution.

- The app.ducx user interface language allows you to define forms, form pages, menu items and other user interface elements for your object classes.
- The purpose of the app.ducx use case language is to define and implement use cases, and provide method implementations for these use cases. Use cases can be implemented in app.ducx expression language or as so-called virtual applications.
- The app.ducx business process language allows you to define the process model for your Cloud App in order to describe and manage workflows.
- The purpose of the app.ducx customization language is to customize and tailor Fabasoft Folio Cloud features provided out-of-the-box to the specific requirements of your Cloud App.
- app.ducx expression language is a distinct domain-specific language of Fabasoft app.ducx. app.ducx expressions can be embedded inline in an `expression` block in other domain-specific languages. app.ducx expression language is processed by the Fabasoft app.ducx compiler and transformed into Fabasoft app.ducx Expressions, which are evaluated at runtime by the Fabasoft Folio Kernel.

In contrast to all the other DSLs of Fabasoft app.ducx, keywords, predefined functions and predefined variables in app.ducx expression language are not case sensitive.

[Faba11a] provides a comprehensive discussion of the syntax and grammar of the Fabasoft app.ducx DSLs, including app.ducx expression language and the query language for search queries.

6.2 Defining the object model

The object model is the first thing you have to define when building a Cloud App.

Using the app.ducx object model language, you can easily define the basic elements that make up the object model:

- object classes
- properties and fields
- enumeration types
- structures

Every object model element in Fabasoft app.ducx, and also the other persistent model elements yet to be presented in the following chapters (e.g. forms and form pages), must be assigned a unique reference (i.e. a programming name for a particular object class or property). These references should follow the reference naming conventions laid out in [Faba11a].

Object model elements may only be defined within an object model block in object model files with a `.ducx-om` extension. The `objmodel` keyword denotes an object model block. It must be followed by the reference of your Cloud App and curly braces.

You can organize the object model elements making up your Cloud App in as many object model files as you wish. However, the skeleton project created for your Cloud App already contains a file named `model.ducx-om`, which is intended to be used for defining the basic object model of your Cloud App, e.g. the `Logbook` and `TripLog` object classes that we will define further down the road.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
}
```

In addition to the `model.ducx-om` file, your Cloud App also contains a file named `app.ducx-om`, which contains the definition of the app object representing your Cloud App. In the chapter “The finishing touches” on page 116, we will discuss the purpose of the app object and what else should be defined in the `app.ducx-om` file. But for now, let’s focus on the `model.ducx-om` file.

Example

app.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import COOATTREDIT@1.1;
  instance App AppFSCLOGBOOK {
    symbol = SymbolAppFSCLOGBOOK;
    appdescription<language, langcontent> = {
      { LANG_ENGLISH, file("resources/LANG_ENGLISH/appdescription.txt") },
      { LANG_GERMAN, file("resources/LANG_GERMAN/appdescription.txt") }
    }
  }
}
```

Note: To create a new object model file for organizing your object model elements, select “File” > “New” > “Fabasoft app.ducx Object Model File”.

6.2.1 Adding the ‘Trip’ structure

First, we’re going to define a data structure for recording all the required trip information according to Table 1 on page 42, i.e. the place, date and time of departure and arrival and so on.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  struct Trip {
    datetime trpdepartureat;
    string trpdepartureplace;
    unsigned float(6,2) trpstartmileage;
    datetime trparrivalat;
    string trpdestinationplace;
    unsigned float(6,2) trpendmileage;
    unsigned float(6,2) trpmileage readonly(ui);
    timespan trpduration readonly(ui);
    TermComponentObject trptype;
    string trppurpose;
    User trpdriver;
    string trpdrivername;
    string trpvehicleid;
    boolean trpcanceled;
  }
}
```

In Fabasoft app.ducx, the `struct` keyword is used to define a compound type composed of members that can have different types.

To represent a single trip from A to B along with all the other required metadata, we defined a data structure called `Trip`, which is composed of 14 properties of various data types.

The basic data types like `string`, `float`, `boolean` and `datetime` largely behave just like in every other programming language with the exception that a `float` can hold up to 16 digits and a `boolean` may also be `null`.

A `timespan` is an integer number storing the number of seconds between two dates. In the example, the `readonly(ui)` property modifier suffix is applied to the `trpduration` property to turn it into a read-only field in the user interface.

An object pointer property is a property pointing to an instance of the object class provided in place of the data type. No explicit keyword is required for defining an object pointer property. Instead, the object class of the objects that shall be referenced by the object pointer property is used as data type.

For instance, in the `trpdriver` property you can select an instance of object class `User`. When you do so, a reference pointing to the selected user object is stored in the `trpdriver` property. Keep in mind that the `trpdriver` property does not store the actual user object itself, but just a pointer pointing to it. If you delete the user object, the pointer becomes invalid and will return `null` when you access it.

You probably noticed that in the example, the `Trip` structure contains two properties for storing driver information, `trpdriver` and `trpdrivername`. The purpose of this is that later on, we will allow the user recording a new trip to pick a user object in the `trpdriver` property, but then we will store only the user's name in the `trpdrivername` property. If the selected user's name is changed later on, the value stored in the `trpdrivername` property will remain unaffected.

Refer to [Faba11a] for a complete listing of all the data types supported by Fabasoft app.ducx as well as for a comprehensive discussion of property modifier suffixes and their effects.

Finally, there's one more very important thing not to forget: As a Cloud App developer you are required to carefully document your source code in English language. To do so, you have to use the Javadoc syntax for documenting all of your object classes, data types, properties, use cases and so on.

For the sake of brevity, we omit these comments in most of the examples in the book. But you have to document each and every element in your source code in order to reach the documentation ratio target of 100 %, so your Cloud App can successfully pass the release process.

The following example demonstrates how to document your source code using Javadoc style comments.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  /**
   * Structure for recording trip information in a trip log
   */
  struct Trip {
    /**
     * Date and time of departure
     */
    datetime trpdepartureat;
    /**
     * Descriptive name of the place of departure, e.g. address or landmark
     */
    string trpdepartureplace;
    /**
     * Odometer reading of the vehicle at the beginning of the trip
     */
    unsigned float(6,2) trpstartmileage;
    …
  }
}
```

Note: [Orac11b] provides a good reference on how to write comments in Javadoc style.

6.2.2 Using terms for the trip type

The `trptype` property of the `Trip` structure is defined as an object pointer property allowing the user to select instances of object class `FSCTERM@1.1001:TermComponentObject`, commonly referred to as "Terms".

Terms can be used to implement category or type selections so a user can choose from one or multiple terms representing a category, type or state, and make a selection.

The benefit over enumerations is that with terms you can allow users to extend the available choices with custom terms (by allowing Cloud users to create custom terms), whereas enumerations cannot be extended by users.

In the chapter "Restricting the selectable trip types" on page 90, we will limit the selectable objects to one of three predefined terms shipped with your Cloud App so users can only choose between "Private", "Business" and "Commute" to describe the type of a particular trip. However, for now, no filter restrictions apply and all the existing terms accessible by users may be selected in the `trptype` property.

6.2.3 Adding software component references

Whenever you either explicitly or implicitly reuse parts of the functionality provided by another software component (e.g. using an object class provided by another software component in one of your object pointer properties), you have to add a reference to this software component.

Therefore, we have to add a reference to software component `FSCTERM@1.1001`, which provides the `TermComponentObject` object class that we used to define the selectable objects for the `trptype` property. To add a reference to software component `FSCTERM@1.1001`, select "Add Reference" from the context menu of the "Software Component References" tree node of your project in Project Explorer. In the "Select Components" dialog box depicted in Figure 27, select the software component `FSCTERM@1.1001` and click "OK". You may also select more than one software component at a time.

In order to be able to use the short reference `TermComponentObject` when referring to the `FSCTERM@1.1001:TermComponentObject` object class in your code, you must add an import declaration for software component `FSCTERM@1.1001` as shown in the example.

For further information on how to manage software component references and import declarations refer to [Faba11a].

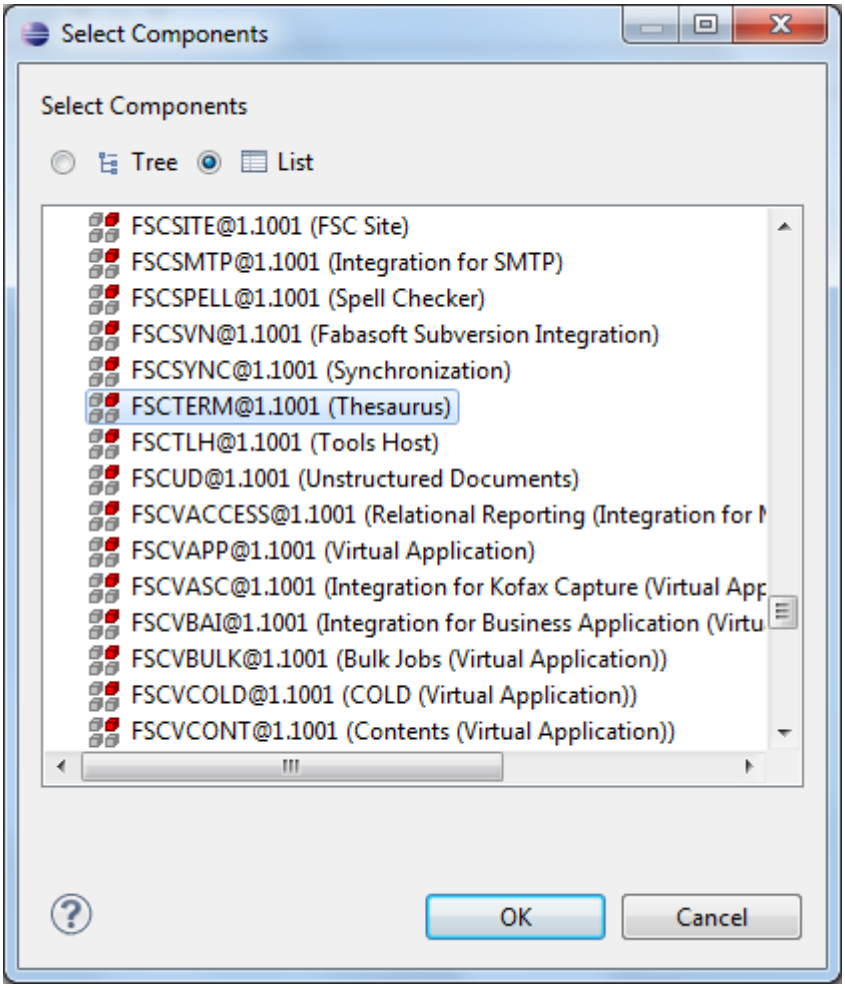


Figure 27: Adding a software component reference

6.2.4 Defining the 'TripLog' object class

After having defined the `Trip` structure, we can now move on and define our first object class.

The `TripLog` object class will be used for storing all the trips of a given month in a property of the `Trip` structure. This property is assigned the reference `trltrips`.

In Fabasoft app.ducx, any property can either be a scalar (meaning that only a single value of the data type assigned to the property can be stored) or a list. Since we want to store all the trips of a month in one trip log object, the `trltrips` property must be defined as a list of `Trip`.

In addition to the `trltrips` property, the trip log also needs a state so that we can distinguish between open trip logs, where the user can still record additional trips, and closed trip logs, where no more changes are permitted.

To model the state of a trip log, we use an enumeration. First, we define the enumeration type `TripLogState` with two enumeration items, `TLS_OPEN` for open and `TLS_CLOSED` for closed trip logs. Then we define a property named `trlstate` of enumeration type `TripLogState` in the `TripLog` object class.

New trip logs should automatically be initialized with the `TLS_OPEN` state. To accomplish that, add the `init` keyword to the definition of the `trlstate` property and set it to `TLS_OPEN`.

Furthermore, we add two `date` properties, `trlfrom` and `trluntil`, to the trip log for storing the departure date of the first non-canceled trip and the arrival date of the last non-canceled trip recorded in the trip log. These properties serve as informational properties only and will be populated automatically.

The properties `trltrips`, `trlstate`, `trlfrom`, and `trluntil` must not be directly changed by the user in the GUI but only through appropriate use cases which we will define later on. Therefore, the `readonly(ui)` property modifier suffix is attached to both properties to prevent users from changing them in the GUI.

We also include another property of the `Trip` data structure in the trip log. The sole purpose of the `trlnewtrip` property is to allow users to enter information for recording a new trip. In a further step, we will implement a mechanism so that the information entered by a user is not saved in the `trlnewtrip` property itself but instead recorded in the `trltrips` property.

Finally, trip logs should be displayed in the tree view when a logbook is expanded. For the instances of an object class to show up in the tree view, set the `compound` keyword to `true`.

The following example illustrates the progress made so far. For the sake of brevity, we will omit parts of the source code already shown in the previous example. Omissions are indicated by a line of dots.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  struct Trip {
    …
  }
  enum TripLogState {
    TLS_OPEN = 1,
    TLS_CLOSED = 2
  }
  class TripLog {
    compound = true;
    TripLogState trlstate readonly(ui) {
      init = TLS_OPEN;
    }
    date trlfrom readonly(ui);
    date trluntil readonly(ui);
    Trip[] trltrips readonly(ui);
    Trip trlnewtrip;
  }
}
```

6.2.5 Defining the ‘Logbook’ object class

The `Logbook` object class is the container for all trip logs belonging to a logbook. Therefore, we derive the `Logbook` object class from the `CompoundObject` object class as it serves as a folder for trip logs.

We need to set the `common` keyword to `true` to make the `Logbook` object class available for selection in the “Create” dialog box so that users can instantiate a logbook either on the home screen, within a folder or within a team room.

Note: The `common` keyword defines whether an object class is available in a “common place”. If you don’t explicitly set `common` to `true` for your object classes then they won’t appear in the “Create” dialog box when a user is creating a new object either on the home screen, within a folder or within a team room.

For object classes that should only be available when creating a new object within a property of one of your own object classes you don't need to set the `common` keyword to `true`. For instance, you can create trip logs within a logbook even though we didn't define the `common` keyword to `true` for the `TripLog` object class.

Logbooks should also appear in the tree view of folders and team rooms. To accomplish that, we need to set the `compound` keyword of the `Logbook` object class to `true`.

Now let's define the properties we need for the `Logbook` object class:

The `logvehicleid` property is a simple `string` storing the unique ID of the vehicle that the logbook is associated with. Most likely, you want to enter the plate numbers of your vehicle for this purpose. We arbitrarily limit the maximum number of characters to 25 and also turn the property into a required field by attaching the `not null` property modifier suffix. For required fields, users must enter values in the GUI in order to be able to save their changes.

In the `logdescription` property, users can enter some descriptive text for their logbooks.

Note: While simple strings are limited to a maximum of 254 characters, string lists (defined using the `string[]` keyword) can store any number of characters – within reason, that is. Don't try to store ten terabytes of text in a string list just for the heck of it.

Now we need to define the `logtriplogs` property, which is required for storing the trip logs belonging to a logbook. We use the `unique` property modifier prefix to indicate that the list of trip logs must not contain duplicate entries.

Lastly, the `child` keyword must be set to `true` for the `logtriplogs` property to indicate the trip logs stored in this property are subordinated to the logbook. The purpose of the `child` keyword is explained in greater detail in the chapter "Things to consider when dealing with team rooms" on page 119.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  struct Trip {
    …
  }
  enum TripLogState {
    …
  }
  class TripLog {
    …
  }
  class Logbook : CompoundObject {
    compound = true;
    common = true;
    string(25) logvehicleid not null;
    string[] logdescription;
    unique TripLog[] logtriplogs {
      child = true;
    }
  }
}
```

6.2.6 Linking logbook and trip logs

Up next, we're going to establish a bi-directional connection between a logbook and its associated trip logs.

Why do we need this?

The `logtriplogs` property of a logbook is a pointer to the trip logs belonging to that logbook, and therefore establishes a link from the logbook to its associated trip logs.

However, in order to find our way back from a trip log to the logbook it belongs to, we need some additional functionality. Otherwise, we wouldn't be able to determine the vehicle ID from the logbook when recording a new trip in a trip log.

First, we need to add an object pointer property to the trip log for pointing to the logbook. In the second step, we will automatically populate the new property `trllogbook` when a trip log is created. The user should not be able to change the property in the GUI, so we add the `readonly(ui)` property modifier suffix to the property definition.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  struct Trip {
    …
  }
  enum TripLogState {
    …
  }
  class TripLog {
    Logbook trllogbook readonly(ui);
    …
  }
  class Logbook : CompoundObject {
    …
  }
}
```

Secondly, using the `link` keyword we link both properties, `trllogbook` of the trip log and `logtriplogs` of the logbook, with each other.

The links ensure that the integrity of the relationship between linked objects is maintained automatically. Whenever a new trip log is added to or removed from the `logtriplogs` property of the logbook, this change is then reflected in the `trllogbook` property of the concerned trip log. Put bluntly, when you add a trip log to the logbook, a pointer to the logbook is stored in the trip log.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  struct Trip {
    …
  }
  enum TripLogState {
    …
  }
  class TripLog {
    Logbook trllogbook readonly(ui) {
      link = logtriplogs;
    }
    …
  }
```

```
  class Logbook : CompoundObject {
    unique TripLog[] logtriplogs {
      link = trllogbook;
      child = true;
    }
    …
  }
}
```

6.2.7 Defining the language strings of your object model elements

Now that we have defined the key object model elements making up your Cloud App, it's time to assign some meaningful names to all the object classes and properties.

To define the strings displayed in the GUI, open the `resources` folder of your Cloud App project. The `resources` folder contains a subfolder for each language you decided to support. Remember when you had to select the check boxes for your Cloud App's supported languages in your development project in Fabasoft Folio Cloud?

The reference of the respective languages is used as name of the language folders. For instance, the reference of the language object representing the English language is `COOSYSTEM@1.1:LANG_ENGLISH`. That's why the folder for English in your Cloud App project is named `LANG_ENGLISH`. Don't try to rename it since the folder name maps to the reference of the language object.

Within the `LANG_ENGLISH` folder, you will find all the multilingual resources for the English language and the same applies for every other supported language.

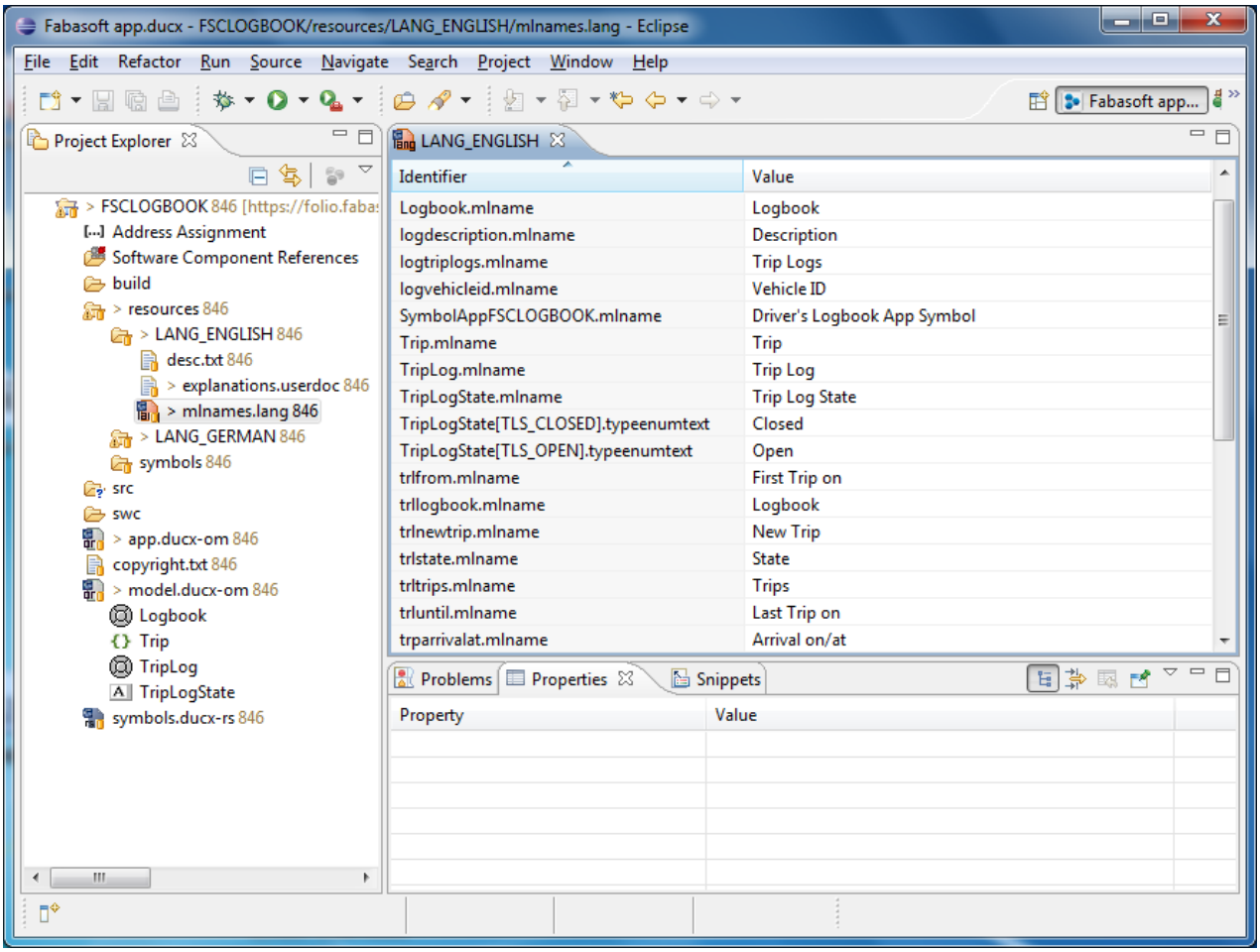


Figure 28: Editing the multilingual names of your object model elements

All multilingual strings for your object model elements, forms and form pages etc. are stored in the `mlnames.lang` file located underneath the language folder. The name `mlnames.lang` is hardcoded, so don't rename this file either.

To define all the English strings for your Cloud App, double-click the `mlnames.lang` file and enter the desired strings for all the entries in the table. Do the same for all other supported languages.

Afterwards, it's always a good idea to clean and recompile your Cloud App project. To do so, select "Clean" from the "Project" menu of Eclipse, select your Cloud App project and click "OK" (see Figure 29). This will recompile the project from scratch to make sure that all changes you've made to the multilingual strings are reflected in the compiler output.

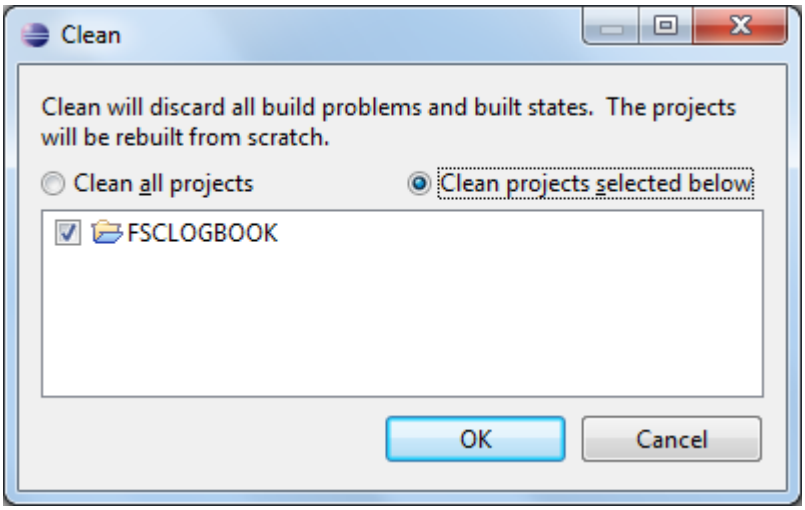


Figure 29: Cleaning the Cloud App project to recompile it from scratch

Instead of defining the multilingual names of your object classes and properties in the `mlnames.lang` file in the respective language folder, you can also use the properties view of Eclipse to enter the language strings for the element selected in the source code (see Figure 30).

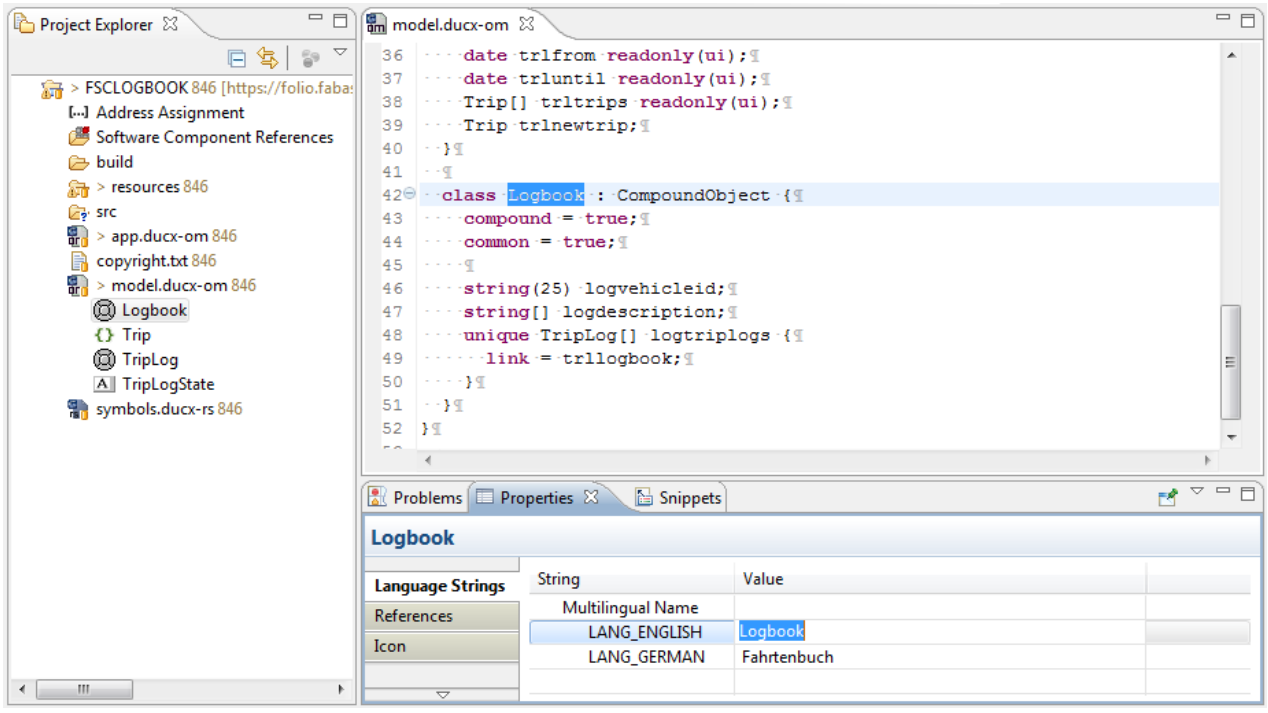


Figure 30: Defining a language string in the properties view

6.3 Defining the symbols

An integral part of a visually compelling Cloud App are the symbols used for instances of your object classes, form pages, menu items and so on.

Using the app.ducx resource language, you can define symbols and resources such as strings and error messages to avoid hard-coded, culture- and language-specific values in your Cloud App.

A resource model block consists of import declarations and resource model elements. The `resources` keyword denotes a resource model block. It must be followed by the reference of your Cloud App and curly braces.

Resource model blocks can only be contained in files with a `.ducx-rs` extension.

Your Cloud App project already contains a `symbols.ducx-rs` file where you can define all the symbols for your Cloud App.

For each symbol, you need to provide six images in the following formats: a GIF image in 16×16 pixels, and PNG images in 16×16, 20×20, 24×24, 256×256 and 512×512 pixels.

The suggested location for image files is the `resources/symbols` folder in your Cloud App project. However, you may freely organize your images in subfolders.

There is already a predefined symbol for your Cloud App in the `symbols.ducx-rs` file with the respective images for the symbol residing in the `resources/symbols` folder. You can use the definition of the `SymbolAppFSCLOGBOOK` symbol as a template for the symbols we need to define for object class `Logbook` and `TripLog`: `SymbolLogbook` and `SymbolTripLog`.

Example

symbols.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  symbol SymbolAppFSCLOGBOOK {
    symbolimages<symbolimageformat, content> = {
      { SF_GIF16,   file("resources/symbols/AppFSCLOGBOOK-GIF16.gif")  },
      { SF_PNG16,   file("resources/symbols/AppFSCLOGBOOK-PNG16.png")  },
      { SF_PNG20,   file("resources/symbols/AppFSCLOGBOOK-PNG20.png")  },
      { SF_PNG24,   file("resources/symbols/AppFSCLOGBOOK-PNG24.png")  },
      { SF_PNG256,  file("resources/symbols/AppFSCLOGBOOK-PNG256.png") },
      { SF_PNG512,  file("resources/symbols/AppFSCLOGBOOK-PNG512.png") }
    }
  }
  symbol SymbolLogbook {
    symbolimages<symbolimageformat, content> = {
      { SF_GIF16,   file("resources/symbols/Logbook-GIF16.gif")  },
      { SF_PNG16,   file("resources/symbols/Logbook-PNG16.png")  },
      { SF_PNG20,   file("resources/symbols/Logbook-PNG20.png")  },
      { SF_PNG24,   file("resources/symbols/Logbook-PNG24.png")  },
      { SF_PNG256,  file("resources/symbols/Logbook-PNG256.png") },
      { SF_PNG512,  file("resources/symbols/Logbook-PNG512.png") }
    }
  }
  symbol SymbolTripLog {
    symbolimages<symbolimageformat, content> = {
      { SF_GIF16,   file("resources/symbols/TripLog-GIF16.gif")  },
      …
      { SF_PNG512,  file("resources/symbols/TripLog-PNG512.png") }
    }
  }
}
```

We suggest using an external image editing program for creating the image files for your symbols. Once you've created the images, import them into your Cloud App project by selecting the `symbols` folder underneath the `resources` folder, as depicted in Figure 31. Then select "Import" from the context menu of the `symbols` folder.

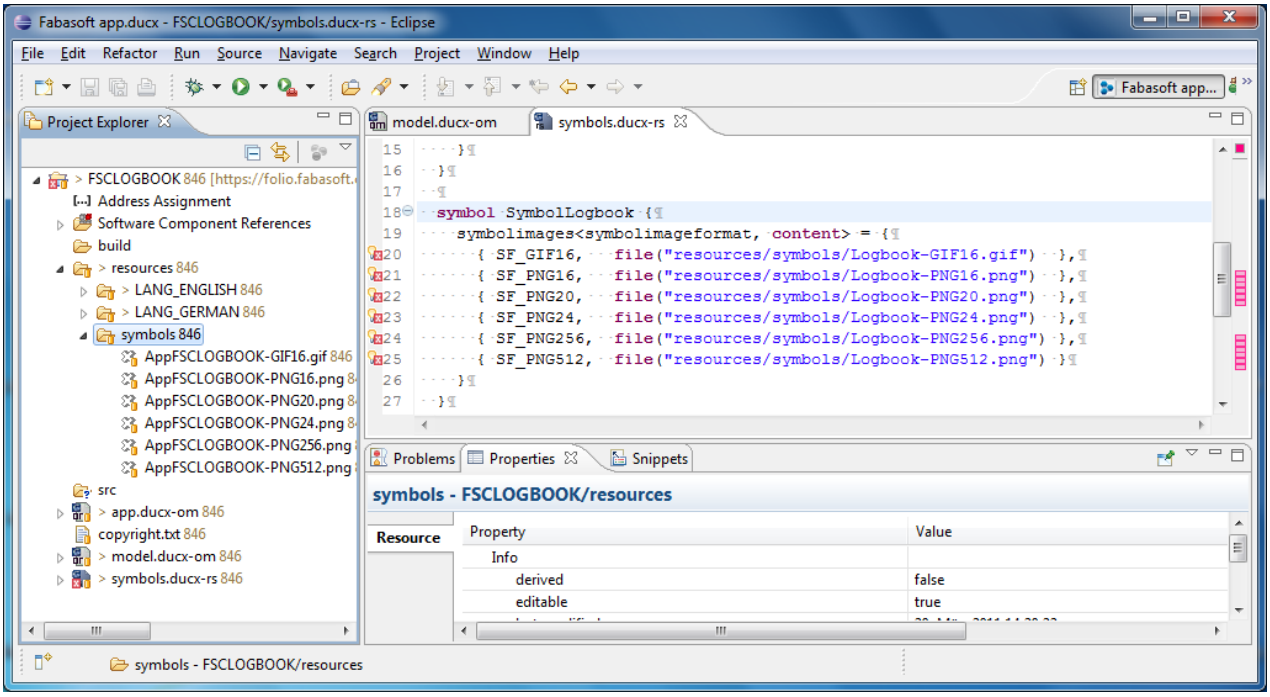


Figure 31: The "symbols" folder contains the image files for the symbols of your Cloud App

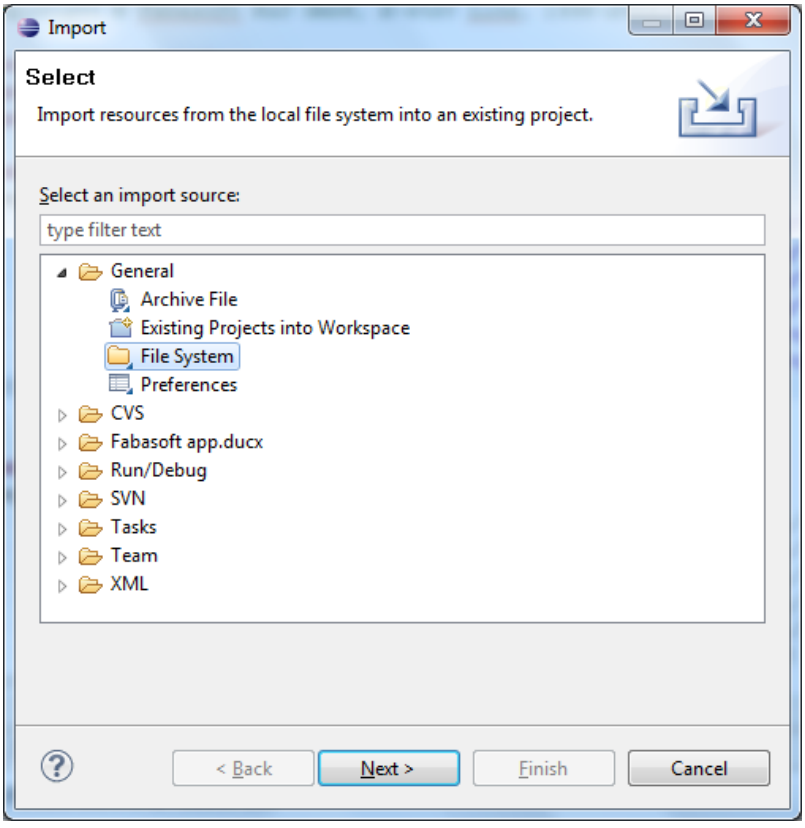


Figure 32: Selecting the "File System" import source

In the dialog box shown in Figure 32, select the “File System” import source from the “General” branch and click “Next”.

In the dialog box depicted in Figure 33, enter the folder name where you saved your images in the *From directory* field and select the image files you want to import. Then click “Finish” to finalize the import of the image files.

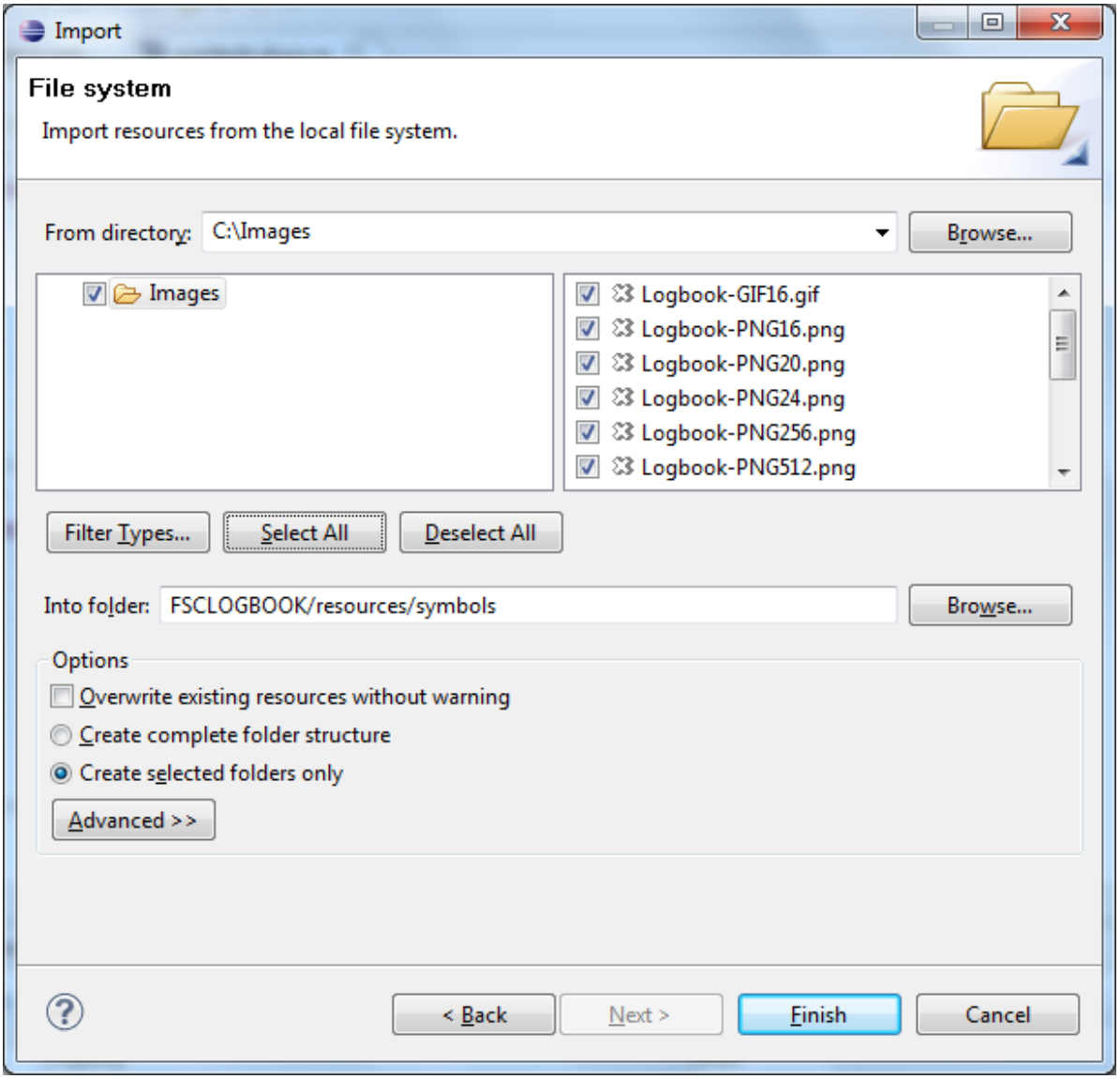


Figure 33: Importing image files from the file system

That’s it!

Now that we have defined our custom symbols for logbooks and trip logs, we’ll have to assign them to the respective object classes in one of the next steps. This will be covered in the chapter “Assigning a symbol to the ‘Logbook” on page 70.

Note: After defining and uploading new symbols into your Cloud App VDE, you need to restart the web services of your Cloud Sandbox for the new symbols to become visible. The reason for this is that the new image files new to be deployed to the image cache of the web service, which is only refreshed after a restart.

For further information on how to restart the web services of your Cloud Sandbox refer to the chapter “Working with the Cloud App VDE” on page 35.

6.4 Designing the forms

With the basic object model along with the required symbols for your object classes in place, we can now tackle the next step: For each object class of your Cloud App, we have to design a set of forms and form pages for displaying the properties.

All the user interface elements for your Cloud App, such as forms, form pages and menu items, are defined using the app.ducx user interface model language.

A user interface model block consists of import declarations and user interface model elements. The `userinterface` keyword denotes a user interface model block. It must be followed by the reference of your Cloud App and curly braces.

User interface model blocks can only be contained in files with a `.ducx-ui` extension.

Just as is the case with all other types of model files, you can have as many `.ducx-ui` files in your Cloud App project as you wish. Usually, it's a good approach to create one for each object class.

6.4.1 Defining a form set for the 'Logbook' object class

It's time to create our first app.ducx user interface file!

From the "File" menu, select "New" and then "Fabasoft app.ducx User Interface File". In the dialog box depicted in Figure 34, enter `logbook.ducx-ui` in the *File name* field and click "Finish".

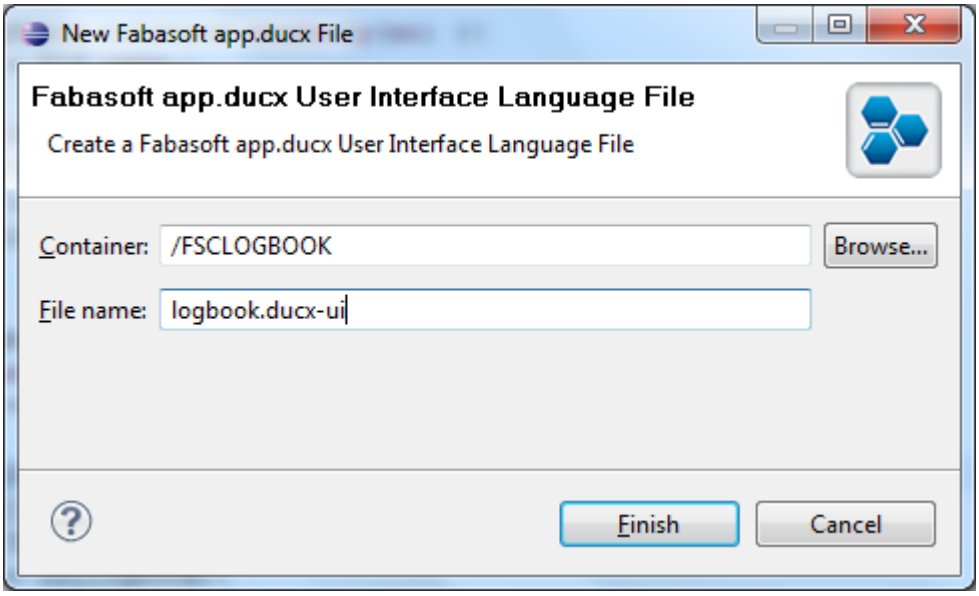


Figure 34: Creating a new app.ducx user interface file

Next, we're going to define a few forms in the `logbook.ducx-ui` file.

Forms serve as "containers" for form pages, which in turn contain the properties displayed in the GUI. In order to be displayed, forms must be bound to specific use cases, which serve as a kind of trigger. When a use case involving a user interface is invoked on an instance of one of your object classes, Fabasoft Folio Cloud tries to locate a matching form for this use case in the form bindings you provided for your object class and displays the matching form. If no match is found, the default form from the base class of your object class is displayed instead.

The most common use cases involving a user interface are listed in Table 2. When creating a new object class, you must also provide form bindings for at least the first four use cases listed in the table.

Use case	Description
COOSYSTEM@1.1:ObjectConstructor	This use case is invoked when a new instance of an object class is created.
COOATTREDIT@1.1:ReadObjectAttributes	This use case is invoked when the properties of an object are read.
COOATTREDIT@1.1:EditObjectAttributes	This use case is invoked when the properties of an object are edited.
COOSEARCH@1.1:SearchObjects	This use case is invoked when the search dialog box is opened.
COODESK@1.1:DisplayOptions	The form assigned to this use case is used for allowing users to select columns when changing the column settings of an object list.
COODESK@1.1:ExploreObject	This use case is invoked when an object is opened in the explore view by selecting the *Explore* menu or by selecting a compound object in the tree.
COODESK@1.1:ExploreTree	The form assigned to this use case defines the object lists shown in the tree view when a compound object is expanded.

Table 2: Use cases for form bindings

So basically, for our `Logbook` object class we have to provide form bindings for the triggers `ObjectConstructor`, `ReadObjectAttributes`, `EditObjectAttributes` and `SearchObjects`. However, as we can map the same form for multiple triggers, we will end up defining just two different forms: a constructor form and another form that is used for the remaining three bindings.

A form is defined using the `form` keyword. The form pages that make up the form can either be defined inside of the `form` block or outside, underneath the user interface model block. To define a form page, the `formpage` keyword is used. Within a form page, a `dataset` block expresses which properties will be displayed on the form page.

By convention, the references of forms should begin with the `Form` prefix. Constructor forms should be prefixed with `ConstructorForm`, and search forms with `SearchForm`. Likewise, the references of form pages should begin with the `Page` prefix.

Our first form, `FormLogbook`, is a simple form comprised of two form pages, `PageLogbook` and `PageLogbookTripLogs`. It should be used for reading and editing the properties of a logbook as well as for defining search restrictions when searching for logbooks.

In the `dataset` block of `PageLogbook`, we explicitly list the properties to be displayed on the form page. In addition to the `logvehicleid` and `logdescription` properties, we also include the `objname` property, which stores the name of the logbook.

The purpose of the second form page, `PageLogbookTripLogs`, is to display the trip logs associated with the logbook. We also use the `symbol` keyword to assign a symbol to the form page. By default, the symbol of the object class is used for the first form page of a form, but using the `symbol` keyword you can assign custom symbols to the remaining form pages.

Example

logbook.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormLogbook {
    formpage PageLogbook {
      dataset {
        objname;
        logvehicleid;
        logdescription;
      }
    }
    formpage PageLogbookTripLogs {
      symbol = SymbolTripLog;
      dataset {
        logtriplogs;
      }
    }
  }
}
```

The second form we need to define is the constructor form for logbooks.

Generally, you want the user to provide values for the most important properties of your object class right when they are creating a new instance of it. However, the constructor form should present only the properties that make sense when creating new objects, while leaving out the ones that don't provide any benefit yet.

Since our requirements state that when creating a logbook, a trip log must automatically be created within the logbook, it doesn't make sense to display the list of trip logs in the constructor form.

The `ConstructorFormLogbook` form is merely reusing the existing `PageLogbook` form page.

Example

logbook.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormLogbook {
    …
  }
  form ConstructorFormLogbook {
    PageLogbook;
  }
}
```

Next, we have to take care of the form binding to ensure that the forms we defined are actually invoked when a new logbook is created or opened for editing. To accomplish this, we need to extend object class `Logbook` with a form binding by adding an `extend class` block to our code.

We do not provide a form binding for `COODESK@1.1:ExploreObject` yet, even though the `Logbook` object class is compound. In this case, the default fallback is that all object lists of the object class are displayed on the right-hand pane when a logbook is selected in the tree view.

Note: In the chapter “Defining the columns for the ‘logtriplogs’ property” on page 75, we will demonstrate how to define a desk form with a form page containing custom column settings for the `logtriplogs` property.

Example

logbook.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormLogbook {
    …
  }
  form ConstructorFormLogbook {
    …
  }
  extend class Logbook {
    forms {
      ObjectConstructor    { ConstructorFormLogbook }
      ReadObjectAttributes { FormLogbook }
      EditObjectAttributes { FormLogbook }
      SearchObjects        { FormLogbook }
    }
  }
}
```

6.4.2 Assigning a symbol to the ‘Logbook’ object class

In the chapter “Defining the symbols” on page 64 we discussed how to define custom symbols for your object classes. However, these symbols won’t be displayed unless we assign them to your object classes.

Within the extend class block for object class `Logbook`, use the `symbol` keyword to assign `SymbolLogbook` to the object class.

Example

logbook.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  …
  extend class Logbook {
    symbol = SymbolLogbook;
    …
  }
}
```

6.4.3 Defining a form set and symbol for the ‘TripLog’ object class

Basically, to define a form set for trip logs we just need to repeat the same steps discussed before.

Create a new Fabasoft app.ducx user interface file named `triplog.ducx-ui` and define a form with the reference `FormTripLog` consisting of just a single form page, `PageTripLog`.

In the `dataset` block of `PageTripLog`, specify the `TripLog` object class. This shortcut allows you to include all properties assigned to the `TripLog` object class instead of having to list them one by one. In the chapter “Layouting form pages using the form designer” on page 71, we will explain how to use the form designer of Fabasoft app.ducx to select the properties that are actually displayed when we define the layout for the form page.

Later on, we will define an automatic name build for trip logs, so there’s no need to include the `objname` property on the `PageTripLog` form page.

Using the `extend class` keyword, we assign the `SymbolTripLog` symbol to the `TripLog` object class as well as the form binding.

Note that we don’t provide a form binding for `COOSYSTEM@1.1:ObjectConstructor`, since users will not have to create trip logs manually but instead will use a wizard to do so – as we will see later on.

Also, we do provide a form binding for `COODESK@1.1:ExploreObject`, which will display `FormTripLog` on the right-hand pane when a trip log is selected in the tree view.

Example

triplog.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormTripLog {
    formpage PageTripLog {
      dataset {
        TripLog;
      }
    }
  }
  extend class TripLog {
    symbol = SymbolTripLog;
    forms {
      ReadObjectAttributes { FormTripLog }
      EditObjectAttributes { FormTripLog }
      SearchObjects        { FormTripLog }
      ExploreObject        { FormTripLog }
    }
  }
}
```

6.4.4 Layouting form pages using the form designer

The form designer of Fabasoft app.ducx (depicted in Figure 35) allows you to define a layout for your form pages using a GUI. To activate the form designer, switch from the *Code* pane to the *Form Pages* pane. The *Palette* contains all properties that are defined in the `dataset` block of the selected form page.

The form designer provides following features:

- The properties can be adjusted within the form page by drag-and-drop.
- Labels and fields can be spanned horizontally or vertically over multiple columns or lines.
- Pressing and holding the `Alt` key allows you to select the label and to adjust it within the field. A label can be positioned left, right, at the top or at the bottom of a field.
- A horizontal rule can be inserted by selecting it from the “Static Controls” block.
- To filter the available properties use the *Filter* field. Clicking “x” deletes the filter.

- Using the context menu, you can assign a control to a property (see [Faba11e]).

If you define a layout for a form page using the form designer, a `layout` block is automatically generated in the source code of the concerned Fabasoft app.ducx user interface file.

Note: Once defined, the `layout` block overrides the `dataset` block. If you don't add a property listed in the `dataset` block to the graphical layout, it will not be displayed in the GUI. However, in order to become available in the palette, a property has to be included in the `dataset` block. Alternatively, if you list the reference of an object class in the `dataset` block, all the properties directly assigned to the object class become available in the palette. This does not include the properties of base classes though, which must be explicitly listed in the `dataset` block.

6.4.4.1 Defining the layout of the "PageTripLog" form page

To define a layout for the `PageTripLog` form page, activate the form designer by switching to the *Form Pages* pane and show the palette by clicking the arrow-shaped "Show Palette" button in the upper right corner.

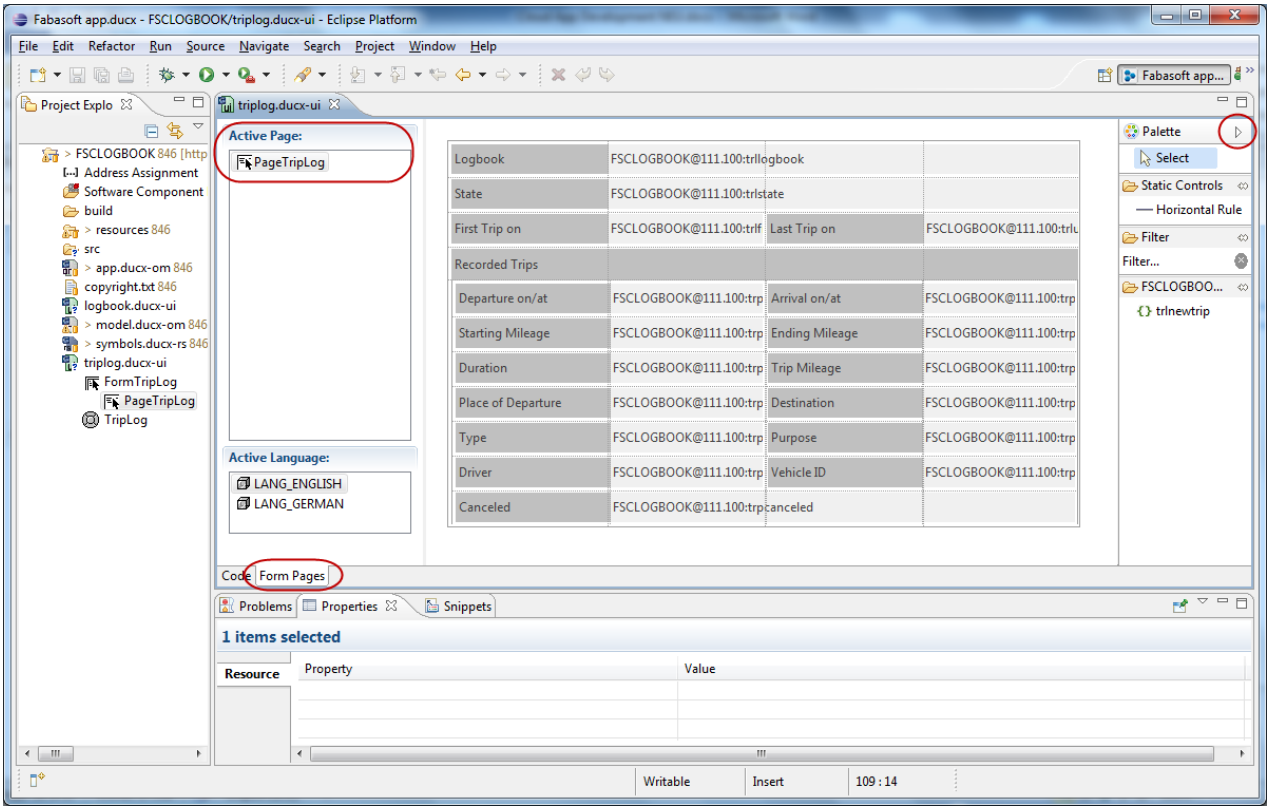


Figure 35: Using the form designer of Fabasoft app.ducx

In the *Active Page* list on the left, select the `PageTripLog` form page. Then click on the `FSCLOGBOOK@111.100:TripLog` category in the palette to expand it.

In the list of available properties, click the `trllogbook` property to select it. Move the mouse pointer over the box on the white canvas and click on it to position the `trllogbook` property on the form page.

Repeat the previous step for all properties except `trlnewtrip`, which will not be included on the `PageTripLog` form page.

To position two properties side-by-side, select the first property, move the mouse pointer over the central anchor point of the left edge of the property and drag it to the right to resize it. Repeat this step for the second property, and then drag the second property next to the first one.

In order to remove a property from the canvas, select it and press the “Delete” key on your keyboard.

6.4.4.2 Modifying the columns of the “Trip” structure

For object lists and structures, you can define the columns or properties displayed for the respective element by defining a so-called “detail layout”.

If you just place a structure (e.g. the `trltrips` property) on a form page without making any modifications, all of the properties that are part of the structure will be displayed on the form page.

However, if you want only a subset of the properties of a structure to be displayed on the form page then you can define a detail layout for the structure.

So let’s go ahead on define a detail layout for the `trltrips` property:

1. If you haven’t done so already, place the `trltrips` property on the form designer canvas
2. Select the `trltrips` property on the canvas
3. In the palette, expand the `FSCLOGBOOK@1.1001:Trip` category
4. Select the `trpdepartureat` property of the `Trip` structure in the palette, point your mouse over the `trltrips` property on the form designer canvas, and drop the `trpdepartureat` property
5. Repeat the previous step for all properties of the `Trip` structure except `trpdriver`

Why did we skip the `trpdriver` property? Because this property will only be used for entering trip data. For displaying recorded trips, it’s of no use as it does not contain a value. When it comes to building the wizard for recording trips, we will explain that in greater detail.

When you’re done with the form page, the final version of the source code for `FormTripLog` should look like the following example.

Example

triplog.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormTripLog {
    formpage PageTripLog {
      dataset {
        TripLog;
      }
      layout {
        // Auto-generated layout block
        row {
          FSCLOGBOOK@111.100:trllogbook {
            colspan = 4;
            labelposition = left;
          }
        }
        row {
          FSCLOGBOOK@111.100:trlstate {
            colspan = 4;
            labelposition = left;
          }
        }
        row {
          FSCLOGBOOK@111.100:trlfrom {
            colspan = 2;
```

```
        labelposition = left;
      }
      FSCLOGBOOK@111.100:trluntil {
        colspan = 2;
        labelposition = left;
      }
    }
    row {
      FSCLOGBOOK@111.100:trltrips {
        detail = layout {
          row {
            FSCLOGBOOK@111.100:trpdepartureat {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trparrivalat {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpstartmileage {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpendmileage {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpduration {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpmileage {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpdepartureplace {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpdestinationplace {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trptype {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trppurpose {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpdrivername {
              colspan = 2;
              labelposition = left;
```

```
                }
                FSCLOGBOOK@111.100:trpvehicleid {
                  colspan = 2;
                  labelposition = left;
                }
              }
              row {
                FSCLOGBOOK@111.100:trpcanceled {
                  colspan = 4;
                  labelposition = left;
                }
              }
            }
            rowspan = 2;
            colspan = 4;
            labelposition = top;
          }
        }
        row {
        }
      }
    }
  }
  extend class TripLog {
    …
  }
}
```

6.4.5 Defining the columns for the ‘logtriplogs’ property

We can also use the form designer to define the default column settings that are shown when an object list is displayed.

For example, when the list of trip logs belonging to a logbook is displayed, it would be nice to display some additional columns besides the name of the respective trip logs.

To accomplish that, we need to add a `layout` block to the `PageLogbookTripLogs` form page. Open the `PageLogbookTripLogs` form page in the form designer, select the `logtriplogs` property, press the “Delete” key to remove it from the form page and then add it to the form page again by selecting it in the palette and placing it on the canvas. This step is necessary to allow for the column settings to become editable.

Select the `logtriplogs` property and then select the `objname` property on the palette and place it inside of the `logtriplogs` property on the canvas. Repeat this step for the `trlstate` and the `trltrips` property.

After saving your changes, the form designer generates a `layout` block for the `PageLogbookTripLogs` form page that will show the `objname` property, the `trlstate` property and the `trltrips` property as separate columns when the `logtriplogs` list is displayed in the property editor.

Since we also want to use the same column settings for the `logtriplogs` list when a logbook is displayed on the desk, we need to define a desk form named `DeskFormTripLog` containing the `PageLogbookTripLogs` form page and provide a form binding for `COODESK@1.1:ExploreObject`, which will display `DeskFormTripLog` on the right-hand pane when a logbook is opened on the desk or selected in the tree view.

Example

logbook.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  …
  form FormLogbook {
    formpage PageLogbook {
      …
    }
    formpage PageLogbookTripLogs {
      symbol = SymbolTripLog;
      dataset {
        logtriplogs;
      }
      layout {
        // Auto-generated layout block
        row {
          FSCLOGBOOK@111.100:logtriplogs {
            detail = layout {
              row {
                COOSYSTEM@1.1:objname {
                  colspan = 4;
                  labelposition = left;
                }
              }
              row {
                FSCLOGBOOK@111.100:trlstate {
                  colspan = 4;
                  labelposition = left;
                }
              }
              row {
                FSCLOGBOOK@111.100:trltrips {
                  rowspan = 2;
                  colspan = 4;
                  labelposition = top;
                }
              }
              row {
              }
            }
            rowspan = 2;
            colspan = 4;
            labelposition = top;
          }
        }
        row {
        }
      }
    }
  }
  …
  form DeskFormLogbook {
    // Reuse the existing "PageLogbookTripLogs" form page
    PageLogbookTripLogs;
  }
```

```
  extend class Logbook {
    forms {
      ObjectConstructor     { ConstructorFormLogbook }
      ReadObjectAttributes  { FormLogbook }
      EditObjectAttributes  { FormLogbook }
      SearchObjects         { FormLogbook }
      ExploreObject         { DeskFormLogbook }
    }
  }
}
```

6.4.6 Beefing up a form page

`layout` blocks generated by the form designer are pretty powerful. And as usual, most of the power remains hidden under the hood.

In the `layout` block of a form page, you can override various settings for properties or define constraints specific to the particular form page.

For instance, you can add validation constraints and user interface change constraints, which we will cover later on in the chapter “Adding a wizard for recording a trip” on page 82.

But for now, we will keep it simple. All we want to accomplish is that the `objname` property becomes a required field where users must enter a value. Moreover, we will assign an HTML editor control to the `logdescription` property so users can apply some formatting when entering description text.

To turn a property into a required field on form page level, add a `mustbedef` expression to the property in the `layout` block. The expression must yield a Boolean return value.

In order to attach the HTML editor control included in Fabasoft Folio Cloud to the `logdescription` property, add a reference to software component `FSCDOX@1.1001` to your Cloud App project (see “Adding software component references” on page 57). Then, write the fully qualified reference of the HTML editor control, `FSCDOX@1.1001:CTRLHtmlEditor`, along with appropriate control parameters in parentheses before the reference of the `logdescription` property.

Refer to [Faba11e] for a detailed description of the controls provided out-of-the-box by Fabasoft Folio Cloud and their respective control parameters.

The following example shows the revamped `PageLogbook` form page.

Example

logbook.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormLogbook {
    formpage PageLogbook {
      dataset {
        objname;
        logvehicleid;
        logdescription;
      }
      layout {
        // Auto-generated layout block
        row {
          COOSYSTEM@1.1:objname {
            colspan = 4;
            labelposition = left;
```

```
            mustbedef = expression {
              return true;
            }
          }
        }
        row {
          FSCLOGBOOK@111.100:logvehicleid {
            colspan = 4;
            labelposition = left;
          }
        }
        row {
          FSCDOX@1.1001:CTRLHtmlEditor(
            "XHTMLDocumentType=FSCDOX@1.1001:XHTMLSelfContained
            NoTableToolbar=true NoNearbar=true DisabledButtons=insertimage;importimage;
            insertresource;createlinkinternal;createexpr;unexpr;togglesource")
            FSCLOGBOOK@111.100:logdescription {
            rowspan = 2;
            colspan = 4;
            labelposition = top;
          }
        }
        row {
        }
      }
    }
    …
  }
  …
}
```

6.5 Committing your changes to the Subversion repository

Safety first! The golden rule when it comes to committing your work to the Subversion repository of Fabasoft Folio Cloud is simple: Do it regularly and often!

The checked in Fabasoft app.ducx project for your Cloud App is what counts at the end of the day. To be more precise, the Fabasoft app.ducx project in the Subversion repository is what will be submitted to Fabasoft once you send your Cloud App into the release process.

Don’t just commit all of your work once you’re completely done with your entire Cloud App. Commit your changes whenever you finish a block of work (e.g. an object class and its properties or a form set). Once committed, you can always roll back to the committed version in case of problems (e.g. data loss due to a crashed hard drive).

To commit your changes to the Subversion repository, select your Cloud App project in Project Explorer, open the context menu and select “Team” > “Commit”. In the dialog box depicted in Figure 36, enter a comment describing the changes that you’ve made in the *Comment* field and click “OK”.

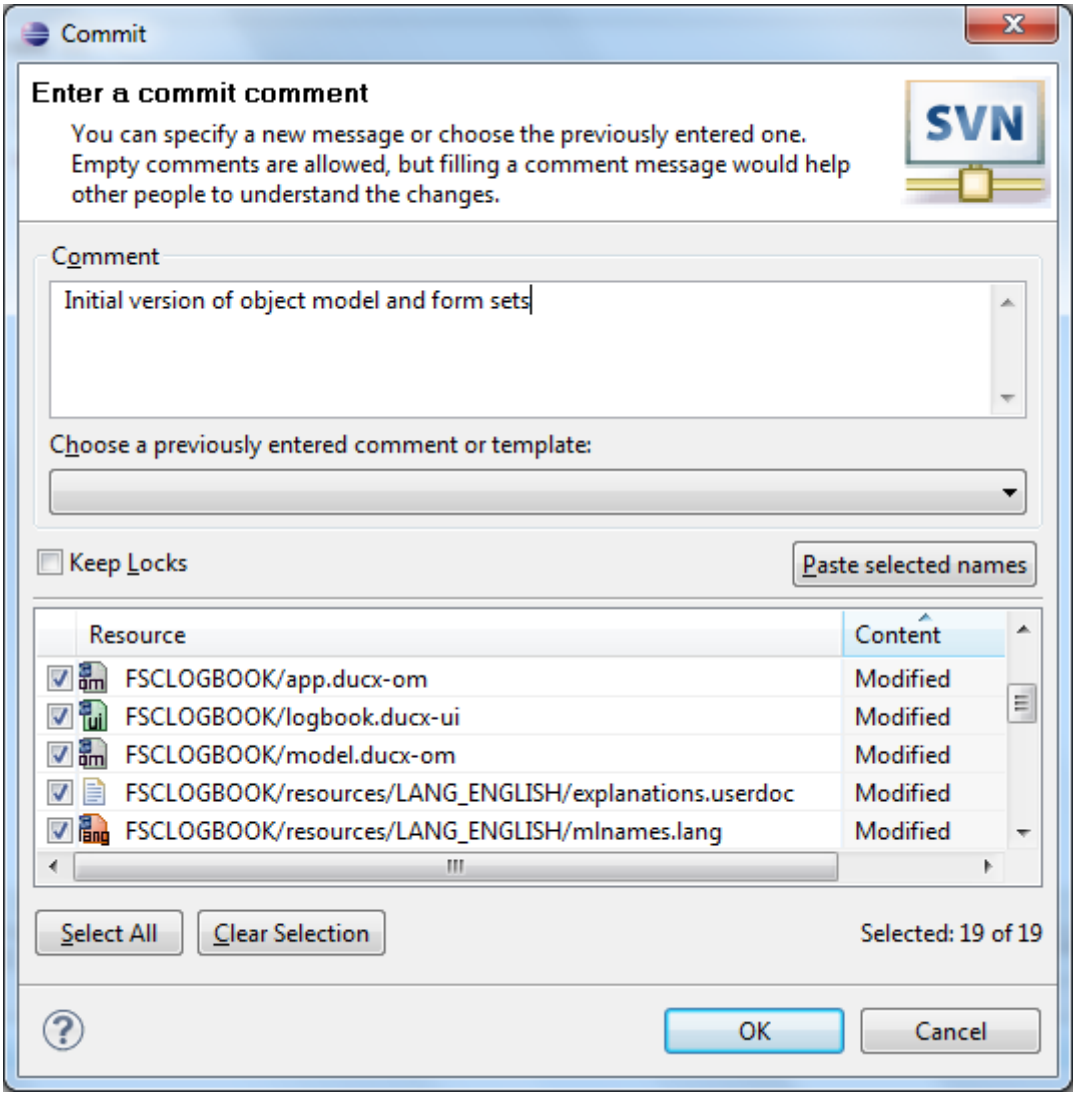


Figure 36: Committing your changes to the Subversion repository

If you are prompted for your credentials, enter your Fabasoft Folio Cloud credentials to proceed.

6.6 Uploading your Cloud App into the Cloud Sandbox

Hey, we've made pretty good progress!

Let's have a look at your Cloud App in your Cloud Sandbox environment so we can actually see what we've accomplished so far!

But before we can see anything, we have to deploy your Cloud App into the Cloud Sandbox.

Whatever you do within Eclipse does not affect the Cloud Sandbox until you actually upload your changes.

6.6.1 Deploying your Cloud App

To upload your Cloud App, you have to create a launch configuration for your Fabasoft app.ducx project in Eclipse. Click "Run Configurations" on the "Run" menu to bring up the dialog box shown in Figure 37.

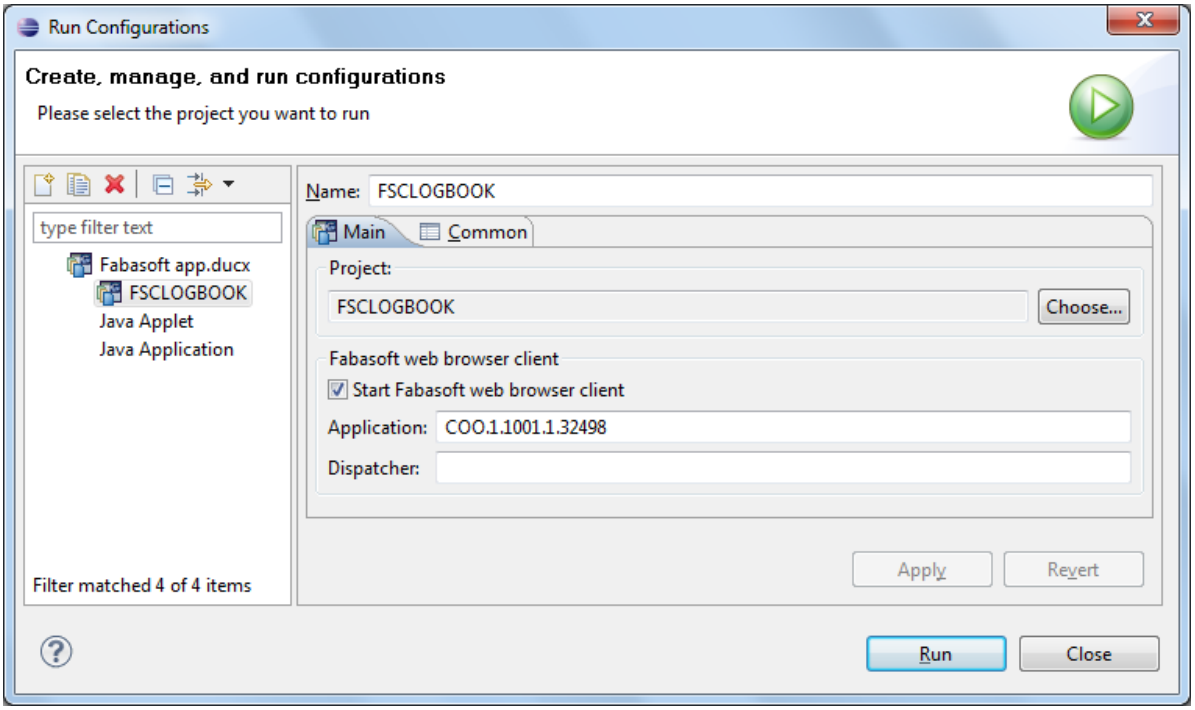


Figure 37: Creating a new launch configuration in Eclipse

In this dialog box, select the “Fabasfot app.ducx section, click the “New launch configuration” symbol and enter a *Name* for the new launch configuration. In addition to this, select the *Project* by clicking “Choose”. Click “Apply” to save your settings, and “Run” to deploy and run your Cloud App project.

Once a Fabasoft app.ducx launch configuration has been created, you can select the existing launch configuration on the “Run as” menu to run your Cloud App project.

6.6.2 Assigning your Cloud App to a test user

After successfully compiling your project and uploading it into your Cloud Sandbox, your browser is opened and you are taken to your Cloud Sandbox where you are requested to log in.

Log in using the “developer” account along with the password you assigned to the Cloud App VDE users (see chapter “Working with the Cloud App VDE” on page 35).

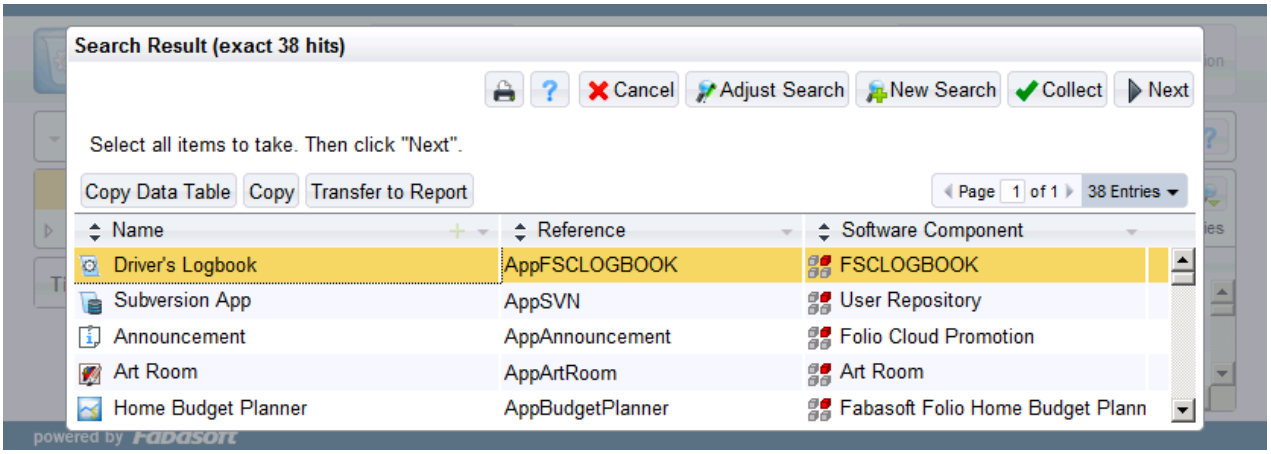


Figure 38: Searching your Cloud App

On your home screen, create an “Administration Tool” and open it. In the administration tool, issue a search for objects of object class “App”. In the search results, select your Cloud App and click “Next” to add it to the administration tool (see Figure 38).

From the context menu of your Cloud App, select “Edit App Users” to bring up the dialog box depicted in Figure 39. In the *Activated for the following user* property, select “Wanda Carney0001” and click “Next”.

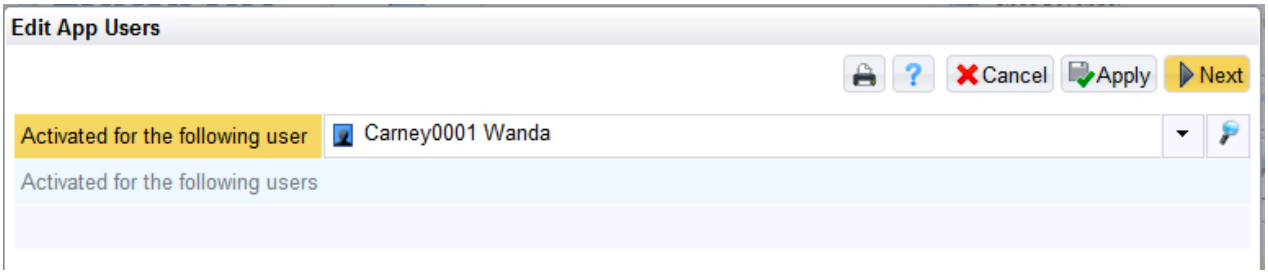


Figure 39: Assigning your Cloud App to a test user

For initial testing, we will use the user account of Wanda Carney. However, using this wizard you can assign a “license” for your Cloud App to any of the test users listed in [Faba11c].

After activating your Cloud App for Wanda Carney, close your current browser session and reconnect to your Cloud Sandbox with the `carney0001` user account.

6.6.3 Your first glimpse of your Cloud App

After logging in to your Cloud Sandbox with the `carney0001` user account, you can go ahead and create your first logbook.

First, create a new folder on the desk of Wanda Carney by clicking the “Create Folder” button. Assign a name to the folder and open it with a double click. Inside the folder, click the “New” button to create a new object, select “Logbook” in the list of available object classes and click “Next”.

In the “Create Logbook” dialog box (see Figure 40), enter a name for your new logbook and enter the plate numbers of your car in the *Vehicle ID* property. Then enter a brief description for the logbook in the *Description* property and click “Next”.

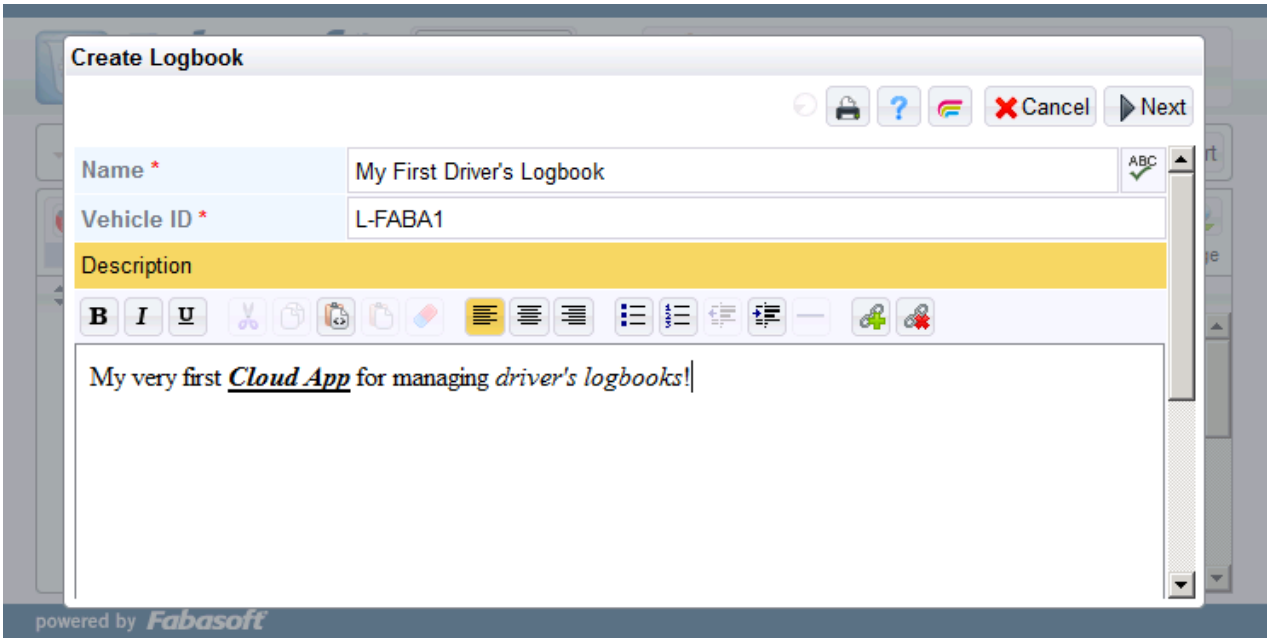


Figure 40: Creating a logbook

If you want, you can also go ahead and open the logbook with a double click and click “New” to create a new trip log in the list of trip logs belonging to the logbook. Then open the properties of the trip log by selecting “Properties” from its context menu.

Well, that’s pretty much what we’ve achieved so far. It’s not the next “Cloud App of the Year” yet, but step by step we’ll get a little closer: In the next chapter we’ll add some cool functionality for recording and canceling trips!

6.7 Implementing the use cases

How about some real coding after all of this forms stuff? Let’s talk about implementing methods! Or rather, what I should say: let’s talk about use cases!

A use case can be implemented on an object class either as a method (using Fabasoft app.ducx expression language) or as a virtual application.

If at any point your use case needs to present a user interface – such as a form or a dialog box – you are required to implement it as a virtual application. This is also the case when you invoke another use case requiring user interaction from your use case.

When no user interaction is required at all, you can (and generally should) implement your use case using Fabasoft app.ducx expression language.

Use cases and virtual applications are defined using the Fabasoft app.ducx use case language in files with a `.ducx-uc` extension.

A use case model block consists of import declarations, transaction variable declarations, and use case model elements. The `usecases` keyword denotes a use case model block. It must be followed by the reference of your Cloud App and curly braces.

6.7.1 Adding a wizard for recording a trip

Now we will implement the centerpiece of the entire Cloud App: the wizard for recording new trips in a trip log.

Using the context menu or a tip, users should be able to invoke the wizard on a trip log, which will then allow them to enter all the required data for a trip. After some validation, this data should be recorded in the `trltips` property of the trip log.

To allow users to enter their trip data, we will display the `trlnewtrip` property in a dialog box of the wizard. When a user clicks “Record Trip”, the data entered in the `trlnewtrip` property will be retrieved to populate the `trltips` property, but the contents of the `trlnewtrip` property will not be stored. Instead, the `trlnewtrip` property will be reinitialized with default values and users can immediately record the next trip without having to reinvoke the wizard.

Figure 41 shows the wizard that we’re going to implement now.

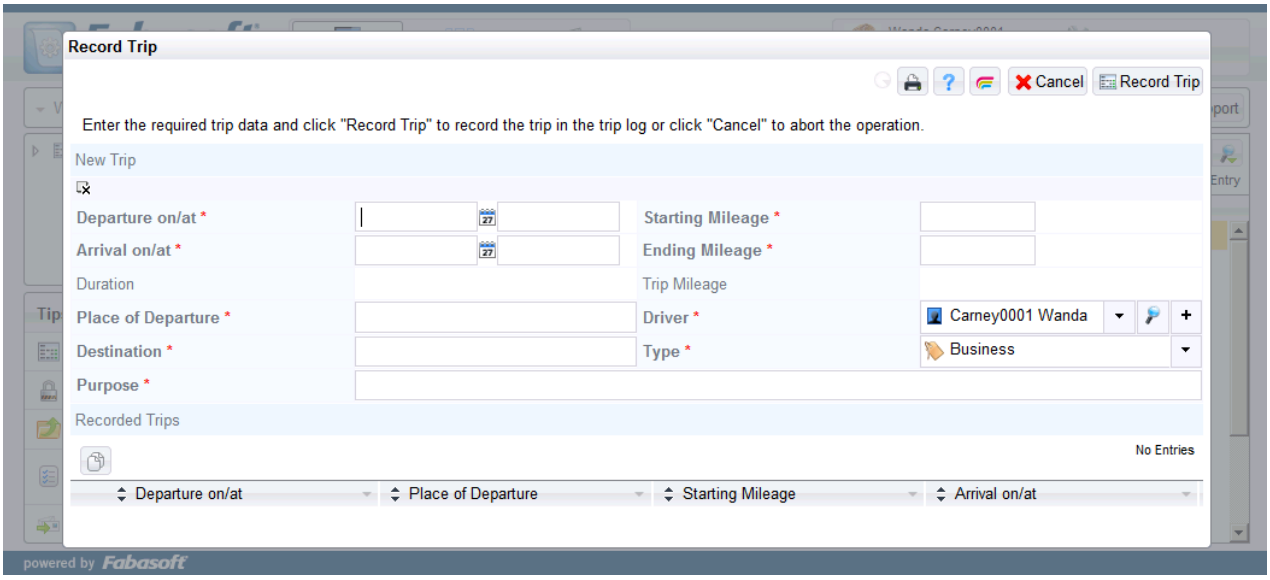


Figure 41: Wizard for recording a new trip

6.7.1.1 Defining a menu use case

To get started, we need to create a new Fabasoft app.ducx use case file named `usecases.ducx-uc`. In the use case file, we define a menu use case with the reference `RecordTripWizard`. In contrast to a simple use case, a menu use case is intended to be invoked using a menu item from the menu, context menu or tips pane.

Note: For menu use cases, the Fabasoft app.ducx compiler implicitly generates a menu item with the same reference as the menu use case and the prefix `Menu`. In our example, a menu item with the reference `MenuRecordTripWizard` is generated automatically, so we don't need to define it manually.

With the `symbol` keyword, we assign a custom symbol to the `RecordTripWizard` use case. This way, the `SymbolRecordTrip` symbol is displayed in the menu item and in the tip entry for invoking the wizard. Basically it's just fuzz, but it makes your Cloud App look much nicer.

Using the `variant` keyword, we tell the Fabasoft app.ducx compiler that we want to implement the `RecordTripWizard` use case in object class `TripLog`.

The `impl = application` statement instructs the Fabasoft app.ducx compiler to create a new virtual application, which will be called whenever the use case is invoked. The `expression` block within the virtual application is its "main function", which is where the virtual application starts execution.

In the `expression` block of the virtual application, we have to check if the trip log is still in "open" state. If it's not, we throw an error, which needs to be defined in a separate Fabasoft app.ducx resource file. In our example, this file is named `errors.ducx-uc`, but you can pick any name you like for your files.

The `errormsg` keyword (allowed within a `resources` block of a `ducx-rs` file) is used to define a custom error. Errors in Fabasoft Folio Cloud are similar to exceptions in Java. The actual (language-specific) error message is specified in the `mlnames.lang` files for each language.

Use the `COOSYSTEM@1.1:Print` action to replace placeholders in an error message with the actual values.

The following example shows what we have accomplished so far. In the next section, we will learn how to define a dialog to interact with users.

For further information on how to define and implement menu use cases and how to raise and format errors, refer to [Faba11a].

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  menu usecase RecordTripWizard on selected or container {
    // Assign a symbol to use case that is displayed in the context menu and tips pane
    symbol = SymbolRecordTrip;
    // Provide an implementation in object class "TripLog"
    variant TripLog {
      // Implement use case "RecordTripWizard" as virtual application
      impl = application {
        // The "main function" of the virtual application
        expression {
          // Check if the trip log is in "open" state
          if (cooobj.trlstate == TLS_OPEN) {
            // TODO: Initialize trip data and display dialog for recording trip
          }
          else {
            // Throw an error if the trip log is not in "open" state
            throw coort.SetError(#ErrTripLogClosed, #ErrTripLogClosed.Print(null,
              cooobj.objname));
          }
        }
        // Open the virtual application in a modal overlay window
        targetwindow = TARGETWINDOW_OVERLAY;
      }
    }
  }
}
```

errors.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  errormsg ErrTripLogClosed;
}
```

mlnames.lang

```
…
ErrTripLogClosed.errtext    Trip log "%1$s" is already closed and must not be changed.
…
```

symbols.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  symbol SymbolRecordTrip {
    symbolimages<symbolimageformat, content> = {
      { SF_GIF16,   file("resources/symbols/RecordTrip-GIF16.gif")  },
      { SF_PNG16,   file("resources/symbols/RecordTrip-PNG16.png")  },
      { SF_PNG20,   file("resources/symbols/RecordTrip-PNG20.png")  },
      { SF_PNG24,   file("resources/symbols/RecordTrip-PNG24.png")  },
      { SF_PNG256,  file("resources/symbols/RecordTrip-PNG256.png") },
      { SF_PNG512,  file("resources/symbols/RecordTrip-PNG512.png") }
    }
  }
}
```

6.7.1.2 Defining a dialog

Alright, let's add the GUI portion to our wizard. After all, every virtual application should have a GUI – otherwise there would be no point in implementing a use case as a virtual application.

Essentially, displaying a GUI in Fabasoft Folio Cloud means showing a form or form page to the user along with some buttons, which are commonly referred to as "branches". A so-called "dialog" serves as a container for the form or form page and the branches.

For our wizard, we add a dialog with the reference `DialogRecordTrip`. In the dialog, we display a form page showing the `trlnewtrip` property where users can enter their trip data. Then we create a new Fabasoft app.ducx user interface file named `wizards.ducx-ui` and define a form page named `PageRecordTrip`, which shows all the properties of the `trlnewtrip` structure except `trpdrivername` and `trpcanceled`.

As mentioned before, `trpdrivername` will be populated based on the user object selected in the `trpdriver` property, and there's no point in displaying the `trpcanceled` property when recording a new trip since this property indicates if a trip was canceled.

Furthermore, notice that we turn all of the properties of the `trlnewtrip` structure into required fields (with the exception of the two read-only properties `trpduration` and `trpmileage`) by adding a `mustbedef` expression to the respective properties in the `layout` block. This way, a user has to provide values for all of the required properties when recording a new trip.

For information purposes, we also display the `trltrips` property on the `PageRecordTrip` form page, which is displaying all the trips recorded in the trip log in read-only mode.

Once the form page has been defined, we can go ahead and define the dialog by adding a dialog block to the virtual application.

Using the `form` keyword, you can assign a form or form page to a dialog. In our example, we assign the `PageRecordTrip` form page to `DialogRecordTrip` dialog.

Next, we need to declare the object on which the dialog is operating. The Fabasoft Folio vApp Engine provides a mechanism for automatically loading and storing property values from and to the target object. The `target` keyword is used to assign a target object to a dialog. Usually, the name of a variable holding the desired target object is specified as the target. Additionally the target can be defined as an app.ducx expression.

For our `DialogRecordTrip` dialog, we assign `cooobj` as the target object. `cooobj` is a special variable representing the "current object", i.e. the object the `RecordTripWizard` use case is invoked on. More specifically, it's the trip log object on which you invoke the context menu to record a new trip.

The `description` keyword allows you to define multilingual information text in the `mlnames.lang` files, which is then displayed in the dialog.

In the example, notice how we use the `<~~>` placeholders as part of the multilingual strings in the `mlnames.lang` files to incorporate expressions into the strings. By invoking the `COOSYSTEM@1.1:Print` action on `StrRecordTrip`, we can merge the multilingual string defined for `StrRecordTrip` into other strings. However, when embedding expression in the `mlnames.lang` files you always have to use the fully qualified reference to address component objects. Only `COOSYSTEM@1.1` is optional.

Why do we include these `<~~>` placeholders in our strings? Well, if you decide to change a string later on then you only have to make the change in one spot.

But why don't we do the same thing for `FSCVENV@1.1001:StrCancel` in the `DialogRecordTrip.description` string then? That's because the string stored in `FSCVENV@1.1001:StrCancel` is prefixed with an ampersand to designate the "C" of "Cancel" as a hotkey, and we definitely don't want to have "&Cancel" in our description string.

In the next step, we have to define the branches for our dialog.

The keyword `cancelbranch` allows you to define a “Cancel” branch for aborting the execution of a virtual application. A `cancelbranch` is set to ignore any user input and is implicitly assigned caption `FSCVENV@1.1001:StrCancel` and symbol `COODESK@1.1:SymbolCancel`. Moreover, the default branch expression `coouser.Cancel()` is implicitly assigned to a `cancelbranch`, which throws an exception to stop the execution of the virtual application.

The purpose of the second branch we add is to record the trip data entered by a user. Using the `branch` keyword, we define a new branch named `BranchRecordTrip` and assign a caption string and a symbol to the branch.

For defining the `StrRecordTrip` caption string, we create a new Fabasoft app.ducx resource file named `strings.ducx-rs`, where we can define a new string object using the `string` keyword.

Remember that in order to pass the review, you must not use any hard-coded string literals in your code. All the multilingual strings used in your Cloud App must be contained in the `mlnames.lang` files. That’s why we define a string object for the caption string in the Fabasoft app.ducx resource file instead of simply assigning a string literal to the `BranchRecordTrip` branch.

Within the `BranchRecordTrip` branch, the `expression` keyword is used to define a branch handler, which is invoked when a user clicks on the branch. However, before we go ahead with defining the code for the branch handler we need to take care of invoking our dialog from the main function of the virtual application.

Expressions that are hosted within a virtual application or within a dialog can make use of the detachment operator `->` to invoke another use case, a virtual application, or a dialog. For invoking a dialog, the detachment operator `->` must be followed by the reference of the dialog to be displayed.

Using the detachment operator, you can invoke any of the dialogs defined in the `application` block of your virtual application. You can also invoke dialogs that have been defined in other applications for reasons of reusability. However, reusing dialogs is strongly discouraged since unlike use cases and virtual applications, they are not self-contained units with a defined interface.

In our wizard, we need to invoke the `DialogRecordTrip` dialog from the expression constituting the main function of our virtual application. However, before we actually invoke the dialog, we want to initialize the `trlnewtrip` structure with some default values by invoking the `InitTrip` use case (which is yet to be defined) on the trip log.

Furthermore, in the branch handler expression of the `BranchRecordTrip` branch, we need to carry out the following steps:

1. Store the trip data: For the recording of the trip data entered by the user in the `trltrips` property we will implement a use case named `RecordTrip` in the `TripLog` object class. We will present the actual implementation of the `RecordTrip` use case later on.
2. Save the changes: In order to save all the changes made, we force a hard commit of the current transaction by invoking the `CommitRoot` action on the trip log.
3. Reinitialize the `trlnewtrip` structure: Based on the data from the last recorded trip and the default values, we will reinitialize the `trlnewtrip` structure by invoking the `InitTrip` use case on the trip log, which is also yet to be defined. This way, the user can instantly record another trip without having to the leave the wizard.

Alright, why do we need this `CommitRoot` and what is it all about?

The idea of our wizard is that a user can record multiple trips one after the other until they cancel out by clicking the “Cancel” branch, which triggers a `coouser.Cancel()`. This action rolls back all the changes in the current transaction and exits the virtual application. In order to make sure that a trip is saved when the user hits “Record Trip”, we force a commit of the current transaction.

The explicit `CommitRoot` is something you usually don’t need in your virtual applications, because when they are exited using a regular branch or a `nextbranch`, the current transaction is automatically committed.

Check out the chapter about virtual applications in [Faba11a] for more information on branches (including the `nextbranch`) and branch handlers.

Also, you may have noticed that we use the term “action” when referring to `CommitRoot`. Well, Fabasoft app.ducx supports different types of methods that can be invoked on an object:

- An action is considered to be the declaration of a private method that is implemented on one or more object classes.
- A use case is considered to be the declaration of a public method. It can provide different implementations on one or more object classes.

So which one should you pick for defining a method, use case or action? Generally, the answer is use case, except for when implementing so-called “get actions”, “set actions” and “display actions”.

Refer to [Faba11a] to get a better understanding of the difference between use cases and actions as well as for a discussion of get actions, set actions and display actions.

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  menu usecase RecordTripWizard on selected or container {
    symbol = SymbolRecordTrip;
    variant TripLog {
      impl = application {
        expression {
          if (cooobj.trlstate == TLS_OPEN) {
            // Initialize "trlnewtrip" structure
            cooobj.InitTrip();
            // Display dialog for recording trip
           ->DialogRecordTrip;
          }
          else {
            throw coort.SetError(#ErrTripLogClosed, #ErrTripLogClosed.Print(null,
              cooobj.objname));
          }
        }
        dialog DialogRecordTrip {
          form = PageRecordTrip;
          target = cooobj;
          description = {}
          cancelbranch;
          branch BranchRecordTrip default {
            caption = StrRecordTrip;
            symbol = SymbolRecordTrip;
            expression {
              // Record the trip data entered by the user in the "trltrips" property
              cooobj.RecordTrip(cooobj.trlnewtrip);
              // Force a commit to save the changes
              coouser.CommitRoot();
              // Reinitialize "trlnewtrip" structure
              cooobj.InitTrip();
            }
          }
        }
        targetwindow = TARGETWINDOW_OVERLAY;
      }
    }
```

```
  }
}
```

strings.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  string StrRecordTrip;
}
```

mlnames.lang

```
…
DialogRecordTrip.description Enter the required trip data and click
"<~#FSCLOGBOOK@111.100:
                             StrRecordTrip.Print()~>" to record the trip in the trip
log or
                             click "Cancel" to abort the operation.
DialogRecordTrip.mlname      <~#FSCLOGBOOK@111.100:StrRecordTrip.Print()~>
StrRecordTrip.string         Record Trip
…
```

wizards.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  formpage PageRecordTrip {
    dataset {
      trlnewtrip;
      trltrips;
    }
    layout {
      // Auto-generated layout block
      row {
        FSCLOGBOOK@111.100:trlnewtrip {
          detail = layout {
            row {
              FSCLOGBOOK@111.100:trpdepartureat {
                colspan = 2;
                labelposition = left;
                mustbedef = expression {
                  return true;
                }
              }
              FSCLOGBOOK@111.100:trpstartmileage {
                colspan = 2;
                labelposition = left;
                mustbedef = expression {
                  return true;
                }
              }
            }
            row {
              FSCLOGBOOK@111.100:trparrivalat {
                colspan = 2;
                labelposition = left;
                mustbedef = expression {
                  return true;
                }
              }
              FSCLOGBOOK@111.100:trpendmileage {
                colspan = 2;
```

```
            labelposition = left;
            mustbedef = expression {
              return true;
            }
          }
        }
        row {
          FSCLOGBOOK@111.100:trpduration {
            colspan = 2;
            labelposition = left;
          }
          FSCLOGBOOK@111.100:trpmileage {
            colspan = 2;
            labelposition = left;
          }
        }
        row {
          FSCLOGBOOK@111.100:trpdepartureplace {
            colspan = 2;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
          }
          FSCLOGBOOK@111.100:trpdriver {
            colspan = 2;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
          }
        }
        row {
          FSCLOGBOOK@111.100:trpdestinationplace {
            colspan = 2;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
          }
          FSCLOGBOOK@111.100:trptype {
            colspan = 2;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
          }
        }
        row {
          FSCLOGBOOK@111.100:trppurpose {
            colspan = 4;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
          }
        }
      }
      rowspan = 2;
      colspan = 4;
      labelposition = top;
    }
  }
  row {
```

```
      }
      row {
        FSCLOGBOOK@111.100:trltrips {
          rowspan = 2;
          colspan = 4;
          labelposition = top;
        }
      }
      row {
      }
    }
  }
}
```

6.7.1.3 Restricting the selectable trip types

In the `trptype` property, users should only be able to select one of three possible trip types: “Private”, “Business” or “Commute”.

To accomplish this task, we need to do three things:

- Define three terms to represent the selectable trip types
- Add a filter constraint (casually referred to as a “filter expression”) to the `trptype` property
- Disable the ability for users to create new terms and to search for other terms in the `trptype` property

The three terms must be defined in a Fabasoft app.ducx object model file. Consequently, we create a new file named `instances.ducx-om` and add definitions for three instances of object class `FSCTERM@1.1001:TermComponentObject`. Also, in the `mlnames.lang` files we assign multilingual names to the three terms.

Next, we go back to the definition of the `trptype` property in the `model.ducx-om` file and add a filter expression to the property returning the list of selectable values.

Finally, we have to make a little change to the `PageRecordTrip` form page in the `wizards.ducx-ui` file, where we need to add a `controlstyle` block to the `trptype` property in the detail layout of the `trlnewtrip` structure.

Using a `controlstyle` block, we can disable the ability for users to create new terms or search for existing terms in the `trptype` property.

For further information on filter constraints and `controlstyle` blocks refer to [Faba11a].

Example

instances.ducx-om

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  instance TermComponentObject TermPrivate {
  }
  instance TermComponentObject TermBusiness {
  }
  instance TermComponentObject TermCommute {
  }
}
```

mlnames.lang

```
…
TermBusiness.mlname          Business
TermCommute.mlname           Commute
TermPrivate.mlname           Private
…
```

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  …
  struct Trip {
    …
    TermComponentObject trptype {
      filter = expression {
        // Filter the selectable terms to "Business", "Commute" and "Private"
        return [#TermBusiness, #TermCommute, #TermPrivate];
      }
    }
    …
  }
  …
}
```

wizards.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  formpage PageRecordTrip {
    …
    layout {
      // Auto-generated layout block
      row {
        FSCLOGBOOK@111.100:trlnewtrip {
          detail = layout {
            …
            row {
              …
              FSCLOGBOOK@111.100:trptype {
                colspan = 2;
                labelposition = left;
                mustbedef = expression {
                  return true;
                }
                // Disable creating and searching for terms
                controlstyle = expression {
                  return [ControlStyle(CTRLSTYLE_DISABLECREATE),
                    ControlStyle(CTRLSTYLE_DISABLESEARCH)];
                }
              }
            }
            …
          }
          …
        }
      }
      …
    }
  }
}
```

6.7.1.4 Implementing the validation constraints

Our wizard still doesn't conduct any validation of the data entered by a user. So let's change that and add some validation constraints for checking the date and time of departure and arrival as well as the mileage information provided by the user!

Validation constraints can either be defined as part of a property definition or in the `layout` block of a form page. Since we only want our validation constraints to be triggered when a new trip is recorded, it's sufficient to add them to the `PageRecordTrip` form page that is displayed by the `RecordTripWizard` virtual application.

Bare in mind that validation constraints – no matter if they are defined at property or form page level – are only executed in the GUI, i.e. when a user is interactively entering data in a dialog.

In the `layout` block of the `PageRecordTrip` form page, we define a total of four validation constraints to cover the following requirements:

- The date and time of departure must be before the date and time of arrival.
- The date and time of departure must be after the date and time of arrival of the last recorded, non-canceled trip, if trips have been recorded in the trip log already.
- The starting mileage must be lower than the ending mileage.
- The starting mileage must not be lower than the ending mileage of the last recorded, non-canceled trip, if trips have been recorded in the trip log already.

The commented example illustrates how to implement these validation constraints. For further information on validation constraints refer to [Faba11a].

Example

wizards.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  formpage PageRecordTrip {
    …
    layout {
      // Auto-generated layout block
      row {
        FSCLOGBOOK@111.100:trlnewtrip {
          detail = layout {
            row {
              FSCLOGBOOK@111.100:trpdepartureat {
                colspan = 2;
                labelposition = left;
                mustbedef = expression {
                  return true;
                }
                // Validation constraint for validating the date and time of departure
                validate = expression {
                  // Get the date and time of departure and arrival entered by the user
                  datetime depdate = cooobj.trlnewtrip.trpdepartureat;
                  datetime arrdate = cooobj.trlnewtrip.trparrivalat;
                  // Get the date and time of arrival of the last recorded,
                  // non-canceled trip
                  datetime lastarrdate = cooobj.trltrips[!trpcanceled][0].
                    trparrivalat;
                  if (depdate != null && lastarrdate != null &&
                    lastarrdate >= depdate) {
```

```
                // Throw an error if the departure date entered by the user is
                // before the arrival date of the last recorded, non-canceled trip
                throw #ErrDepartureBeforeLastArrival;
              }
              else if (depdate != null && arrdate != null && arrdate <= depdate) {
                // Throw an error if the arrival date entered by the user is before
                // the departure date
                throw #ErrArrivalBeforeDeparture;
              }
              else {
                // Return "true" to indicate that the validation was successful
                return true;
              }
            }
          }
          FSCLOGBOOK@111.100:trpstartmileage {
            colspan = 2;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
            // Validation constraint for validating the starting mileage
            validate = expression {
              // Get the starting and ending mileage entered by the user
              float start = cooobj.trlnewtrip.trpstartmileage;
              float end = cooobj.trlnewtrip.trpendmileage;
              // Get the ending mileage of the last recorded, non-canceled trip
              float lastend = cooobj.trltrips[!trpcanceled][0].
                trpendmileage;
              if (start != null && lastend != null && lastend > start) {
                // Throw an error if the starting mileage entered by the user is
                // lower than the ending mileage of the last recorded,
                // non-canceled trip
                throw #ErrStartMileageLastEndMileage;
              }
              else if (start != null && end != null && start >= end) {
                // Throw an error if the ending mileage entered by the user is
                // lower than the starting mileage
                throw #ErrStartMileageEndMileage;
              }
              else {
                // Return "true" to indicate that the validation was successful
                return true;
              }
            }
          }
        }
        row {
          FSCLOGBOOK@111.100:trparrivalat {
            colspan = 2;
            labelposition = left;
            mustbedef = expression {
              return true;
            }
            // Validation constraint for validating the date and time of arrival
            validate = expression {
              // Get the date and time of departure and arrival entered by the user
              datetime depdate = cooobj.trlnewtrip.trpdepartureat;
              datetime arrdate = cooobj.trlnewtrip.trparrivalat;
              if (arrdate != null && depdate != null && arrdate <= depdate) {
                // Throw an error if the arrival date entered by the user is before
                // the departure date
                throw #ErrArrivalBeforeDeparture;
              }
```

```
                    else {
                      // Return "true" to indicate that the validation was successful
                      return true;
                    }
                  }
                }
                FSCLOGBOOK@111.100:trpendmileage {
                  colspan = 2;
                  labelposition = left;
                  mustbedef = expression {
                    return true;
                  }
                  // Validation constraint for validating the ending mileage
                  validate = expression {
                    // Get the starting and ending mileage entered by the user
                    float start = cooobj.trlnewtrip.trpstartmileage;
                    float end = cooobj.trlnewtrip.trpendmileage;
                    if (start != null && end != null && start >= end) {
                      // Throw an error if the ending mileage entered by the user is
                      // lower than the starting mileage
                      throw #ErrStartMileageEndMileage;
                    }
                    else {
                      // Return "true" to indicate that the validation was successful
                      return true;
                    }
                  }
                }
              }
              …
            }
            …
          }
        }
        …
      }
    }
}
```

errors.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  errormsg ErrDepartureBeforeLastArrival;
  errormsg ErrArrivalBeforeDeparture;
  errormsg ErrStartMileageLastEndMileage;
  errormsg ErrStartMileageEndMileage;
}
```

mlnames.lang

```
…
ErrDepartureBeforeLastArrival.errtext   Departure must be before arrival.
ErrArrivalBeforeDeparture.errtext       Departure must be after arrival of the last
                                        recorded, non-canceled trip.
ErrStartMileageEndMileage.errtext       Starting mileage must be lower than
                                        ending mileage.
ErrStartMileageLastEndMileage.errtext   Starting mileage must not be lower than ending
                                        mileage of the last recorded, non-canceled
                                        trip.
…
```

6.7.1.5 Implementing the user interface change constraints

Now we'll focus on user interface change constraints, casually referred to as "UI change handlers". These handy guys are invoked when a user changes a property on a form page by entering or selecting a value.

We're going to use them to instantly calculate the duration of the trip and the trip mileage as users enter the required trip data.

Just like validation constraints, user interface change constraints can either be implemented at property or form page level. And once again, we decide to implement them at form page level, because we only want our calculation voodoo to happen in our `RecordTripWizard` virtual application.

The good thing is that the user interface change constraint we need to define for the `trparrivalat` property is exactly identical to the one for the `trpdepartureat` property, and the one for the `trpendmileage` property is identical to the expression for the `trpstartmileage` property. So we'll omit the `trparrivalat` and `trpendmileage` properties in the following example. However, you still have to go the whole nine yards and define all four of the user interface change constraints.

Example

wizards.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  formpage PageRecordTrip {
    …
    layout {
      // Auto-generated layout block
      row {
        FSCLOGBOOK@111.100:trlnewtrip {
          detail = layout {
            row {
              FSCLOGBOOK@111.100:trpdepartureat {
                colspan = 2;
                labelposition = left;
                mustbedef = expression {
                  return true;
                }
                validate = expression {
                  …
                }
                // User interface change constraint for updating the trip
                // duration when the date and time of departure is changed
                change = expression {
                  // Get the date and time of departure and arrival entered by the user
                  datetime depdate = cooobj.trlnewtrip.trpdepartureat;
                  datetime arrdate = cooobj.trlnewtrip.trparrivalat;
                  // Check if departure and arrival date are valid, and if the arrival
                  // date is after the departure date
                  if (arrdate != null && depdate != null && arrdate > depdate) {
                    // Calculate and store the duration of the trip
                    cooobj.trlnewtrip.trpduration = arrdate - depdate;
                  }
                  else {
                    // Empty duration property in case of incomplete or invalid data
                    cooobj.trlnewtrip.trpduration = null;
                  }
                }
              }
```

```
            FSCLOGBOOK@111.100:trpstartmileage {
              colspan = 2;
              labelposition = left;
              mustbedef = expression {
                return true;
              }
              validate = expression {
                …
              }
              // User interface change constraint for updating the total mileage
              // when the starting mileage is changed
              change = expression {
                // Get the starting and ending mileage entered by the user
                float start = cooobj.trlnewtrip.trpstartmileage;
                float end = cooobj.trlnewtrip.trpendmileage;
                // Check if starting and ending mileage are valid, and if the
                // ending mileage is higher than the starting mileage
                if (start != null && end != null && end > start) {
                  // Calculate and store the total mileage of the trip
                  cooobj.trlnewtrip.trpmileage = end - start;
                }
                else {
                  // Empty the trip mileage property is case of incomplete or
                  // invalid data
                  cooobj.trlnewtrip.trpmileage = null;
                }
              }
            }
          }
          …
        }
        …
      }
    }
  }
}
```

6.7.1.6 Implementing the “InitTrip” use case

Let’s add some more bells and whistles to our wizard and populate the `trlnewtrip` structure with initialization values to make it more convenient for users to fill out the required properties.

The following should be accomplished by the implementation of the `InitTrip` use case:

- Initialize the `trpstartmileage` property with the value of the `trpendmileage` property of the most recent, non-canceled recorded trip.
- Initialize the `trpdriver` property with the current user.
- Initialize the `trptype` property with the “Business” term.

The `RecordTripWizard` virtual application is already invoking the `InitTrip` use case before showing the `DialogRecordTrip` dialog, so all that’s left to do is to provide the actual implementation of `InitTrip` as follows in the example.

Note: Whenever you plan on changing an object using expression language, make sure you lock it first by invoking the `COOSYSTEM@1.1:ObjectLock` action on the object. There are a few exceptions where you don’t need to lock the object before you change it, because Fabasoft Folio Cloud already locks it on your behalf (e.g. in the object constructor, a set action or a user interface change constraint). However, generally it’s better to play it safe and place a lock on an object before changing it.

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  menu usecase RecordTripWizard on selected or container {
    …
  }
  usecase InitTrip() {
    variant TripLog {
      impl = expression {
        // Lock the trip log before making any changes
        cooobj.ObjectLock(true, true);
        // Initialize the "trlnewtrip" structure
        cooobj.trlnewtrip = {
          // Retrieve the last recorded, non-canceled trip from the "trltrips"
          // property and get the ending mileage from it, then write it into the
          // "trpstartmileage"property of the "trlnewtrip" structure
          trpstartmileage = cooobj.trltrips[!trpcanceled][0].trpendmileage,
          // Initialize the "trpdriver" property of the "trlnewtrip" structure with
          // the current user, which is always available in the "coouser" variable
          trpdriver = coouser,
          // Initialize the "trptype" property of the "trlnewtrip" structure with the
          // term representing "Business" to indicate that it is a business trip
          trptype = #TermBusiness
        };
      }
    }
  }
}
```

6.7.1.7 Implementing the “RecordTrip” use case

The purpose of the `RecordTrip` use case is to store the trip information passed to the `trip` parameter in the `trltrips` property of the trip log.

We add some level of protection against incomplete trip information by checking if all the required properties are populated with a value before performing the same checks also carried out by the validation constraints.

Then, for reasons of better usability, we add the most recent trip at the beginning of the list of recorded trips.

Instead of storing the object pointer referencing the selected user representing the driver, we store their name as a string in the `trpdrivername` property. This guarantees that the values stored in the trip log will not change once recorded, even if the driver is renamed afterwards.

The following example shows the full source code of the `RecordTrip` use case.

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  menu usecase RecordTripWizard on selected or container {
```

```
  …
}
usecase InitTrip() {
  …
}
usecase RecordTrip(Trip trip) {
  variant TripLog {
    impl = expression {
      // Lock the trip log before making any changes
      cooobj.ObjectLock(true, true);
      // Check if all the required fields are populated
      if (trip.trparrivalat != null && trip.trpdepartureat != null &&
        trip.trppurpose != null && trip.trpstartmileage != null &&
        trip.trpendmileage != null && trip.trpdriver != null &&
        trip.trptype != null && trip.trpdepartureplace != null &&
        trip.trpdestinationplace != null) {
        // Retrieve the last recorded, non-canceled trip and get the date and time of
        // arrival as well as the ending mileage from it
        Trip lasttrip = cooobj.trltrips[!trpcanceled][0];
        datetime lastarrdate = lasttrip.trparrivalat;
        float lastend = lasttrip.trpendmileage;
        if (trip.trpdepartureat != null && lastarrdate != null &&
          lastarrdate >= trip.trpdepartureat) {
          // Throw an error if the departure date entered by the user is before
          // the arrival date of the last recorded, non-canceled trip
          throw #ErrDepartureBeforeLastArrival;
        }
        else if (trip.trparrivalat <= trip.trpdepartureat) {
          // Throw an error if the arrival date entered by the user is before
          // the departure date
          throw #ErrArrivalBeforeDeparture;
        }
        else if (lastend > trip.trpstartmileage) {
          // Throw an error if the starting mileage entered by the user is lower
          // than the ending mileage of the last recorded, non-canceled trip
          throw #ErrStartMileageLastEndMileage;
        }
        else if (trip.trpstartmileage >= trip.trpendmileage) {
          // Throw an error if the ending mileage entered by the user is lower
          // than the starting mileage
          throw #ErrStartMileageEndMileage;
        }
        // Record the trip in the trip log if all the validations succeed
        else {
          if (trip.trpduration == null) {
            // Calculate trip duration if it is not set already
            trip.trpduration = trip.trparrivalat - trip.trpdepartureat;
          }
          if (trip.trpmileage == null) {
            // Calculate trip mileage if it is not set already
            trip.trpmileage = trip.trpendmileage - trip.trpstartmileage;
          }
          // Store the driver's name as string in the "trpdrivername" property and
          // empty the "trpdriver" property
          trip.trpdrivername = trip.trpdriver.objname;
          trip.trpdriver = null;
          // Store the vehicle ID retrieved from the logbook
          trip.trpvehicleid = cooobj.trllogbook.logvehicleid;
          // Record the new trip at the beginning of the "trltrips" list
          cooobj.trltrips = trip + cooobj.trltrips;
        }
      }
      else {
        // Throw an error if incomplete trip information has been provided
```

```
          throw #ErrIncompleteTripInfo;
        }
      }
    }
  }
}
```

errors.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  errormsg ErrIncompleteTripInfo;
}
```

mlnames.lang

```
…
ErrIncompleteTripInfo.errtext           Incomplete trip information entered.
…
```

6.7.1.8 Discarding the values in the “trlnewtrip” property

What happens with the values in the `trlnewtrip` property when recording a new trip?

Well, since we don’t do anything special with them, the values entered by the user in the `trlnewtrip` structure are actually stored in the trip log object.

However, it doesn’t make sense to store data in the `trlnewtrip` structure as the only purpose of it is to allow users to enter values that should eventually be recorded in the `trltrips` property.

Therefore, we need to add a set action to empty the `trlnewtrip` structure before it’s persisted. Moreover, the `trlnewtrip` structure should also not be searchable, so we also have to add a so-called “search action”.

We can use the predefined search action `COOSYSTEM@1.1:AttrSearchNotPossible` to prevent searching in the `trlnewtrip` structure, but we’ll have to implement our own set action for emptying the `trlnewtrip` structure.

The following example demonstrates how to implement our requirements.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  struct Trip {
    …
  }
  enum TripLogState {
    …
  }
  class TripLog {
    …
    Trip trlnewtrip {
      // Add "AttrNewTripSet" as set action
      set = AttrNewTripSet;
      // Add "AttrSearchNotPossible" as search action
      search = AttrSearchNotPossible;
    }
  }
  class Logbook : CompoundObject {
```

```
    …
  }
}
```

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  menu usecase RecordTripWizard on selected or container {
    …
  }
  usecase InitTrip() {
    …
  }
  usecase RecordTrip(Trip trip) {
    …
  }
  // Define a set action for emptying a property so all the values entered by the user
  // in the property are discarded and nothing is stored
  AttrNewTripSet(parameters as AttrSetPrototype) {
    variant TripLog {
      impl = expression {
        // Set the "value" parameter to "null" to discard all the entered values
        value = null;
      }
    }
  }
}
```

6.7.1.9 Adding the wizard to the context menu and tips pane

So we've spent the last couple of pages describing how to implement this wizard for the recording of trips. But how are we going to invoke it? Well, either using the context menu or using the tips pane of the trip log.

In the next step, we'll add the menu item implicitly generated by the menu use case described in the chapter "Defining a menu use case" on page 83 to both the context menu and the tips pane of the trip log.

The easiest way to do this is to use predefined expansion points as illustrated by the example presented in this chapter.

Note: Using a `condition` block, you can define a precondition in Fabasoft app.ducx expression language that must be fulfilled for the menu items listed in the `entries` block to be included in the context menu or tips pane.

Example

triplog.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  …
  extend menu MenuRecordTripWizard {
    // Include the "long text" for the menu item, which is displayed in the tips pane,
    // in the "mlnames.lang" files
    menulongtext = {}
  }
  extend class TripLog {
```

```
    …
    menus {
      default = {
        expansions {
          // The "COODESK@1.1:MenuContextExpansion" expansion point allows you to add
          // custom menu items to the context menu of the instances of an object class
          MenuContextExpansion {
            // Only display the "Record Trip" menu in the context menu if the trip log
            // is in "open" state
            condition = expression {
              cooobj.trlstate == TLS_OPEN;
            }
            entries = {
              MenuRecordTripWizard
            }
          }
          // The "COODESK@1.1:MenuTaskPaneSelectedExpansion" expansion point allows you
          // to add custom menu items to the tips pane of instances of an object class
          TaskPaneSelectedExpansion {
            // Only display the "Record Trip" menu in the tips pane if the trip log
            // is in "open" state
            condition = expression {
              cooobj.trlstate == TLS_OPEN;
            }
            entries = {
              MenuRecordTripWizard
            }
          }
        }
      }
    }
  }
}
```

mlnames.lang

```
…
MenuRecordTripWizard.menulongtext       Record a new trip in the selected trip logs
MenuRecordTripWizard.menustatetext      Records new trips in selected trip logs
MenuRecordTripWizard.mlname             Record Trip
…
```

In addition to the expansions for the default menu binding, you can also define scope-specific menu bindings where the current team room's type (represented by the corresponding app object) is used as the current scope.

For example, within a team room of type "Family Room", the `FSCFOLIOCLOUDFAMILY@1.1001:AppFamilyRoom` app object is used as the current scope. If you define menu expansions for the `FSCFOLIOCLOUDFAMILY@1.1001:AppFamilyRoom` scope, these menus will only be displayed when the user is working on a logbook within a team room of type "Family Room".

For further information on scope-specific menu bindings refer to [Faba11a].

6.7.2 Adding a wizard for canceling a trip

Here's the good news: The hardest part of our work has been done, and from hereon in on it's only going to get easier! But there are still a few things that need to be taken care of. A wizard for canceling recorded trips, for instance.

Our requirements state that it must be possible for a user to cancel the last recorded, non-canceled trip of a trip log. This feature should resemble what you would do by crossing out the last line in a paper logbook, if it has been recorded in error.

Figure 42 shows the final product as we envisage it.

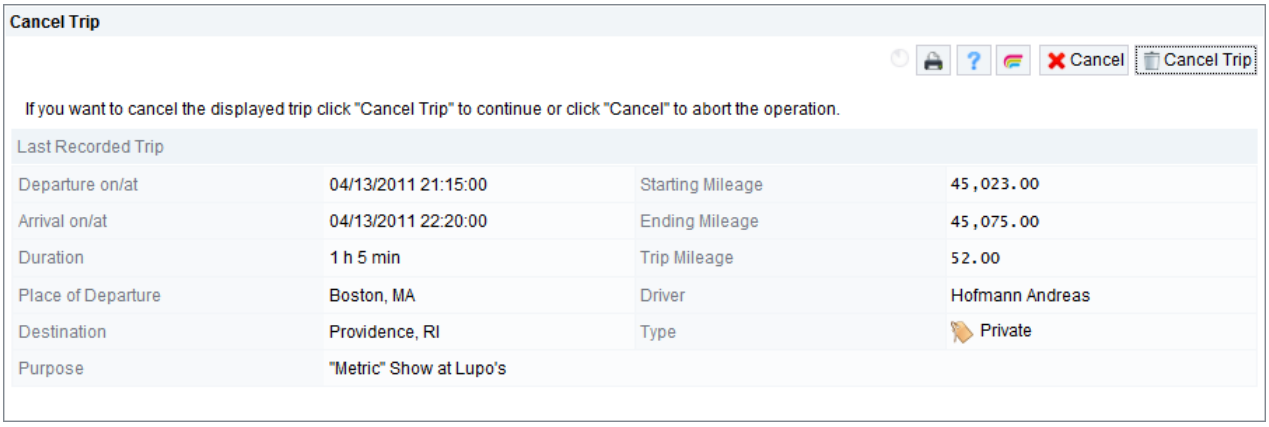


Figure 42: Wizard for canceling recorded trips

Here's an outline of the tasks we need to complete for our trip cancelation wizard:

- First, we have to define another menu use case named `CancelTripWizard`.
- The menu use case needs to invoke a dialog (going by the name of `DialogCancelTrip`), which will display a form page with the reference `PageCancelTrip`.
- We must define another error message, `ErrNoTripsToCancel`, which will be thrown if the wizard is invoked but there are no recorded trips available to cancel.
- Also, we need a new string for the caption of the "Cancel Trip" branch of the dialog, which will be assigned the reference `StrCancelTrip`.
- We will use the form designer to rename the `trlnewtrip` property to "Last Recorded Trip" on the `PageCancelTrip` form page.
- Finally, we will add our menu item for invoking the wizard to the context menu and tips pane expansion points of the `TripLog` object class.

Since you are already a battle-hardened Fabasoft app.ducx veteran by now, we'll do it all at once. Let the comments in the example guide you as they explain the rationale behind our code.

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  …
  menu usecase CancelTripWizard on selected or container {
    symbol = SymbolCancel;
    variant TripLog {
      impl = application {
        // Define application global variables for the whole list of recorded trips and
        // the index of the entry we want to cancel. Application global variables are
        // kept in memory throughout the entire virtual application, whereas "regular"
        // variables are not transported from dialog to dialog. Refer to [Faba11a] for
        // further information on application global variables.
        Trip[] triplist;
        integer tripidx;
        expression {
          // Check if the trip log is in "open" state
          if (cooobj.trlstate == TLS_OPEN) {
```

```
            // Lock the trip log before making any changes
            cooobj.ObjectLock(true, true);
            // Get the list of all recorded trips from the trip log and count the
            // number of entries
            triplist = cooobj.trltrips;
            integer tripcount = count(triplist);
            // Find the index of the last recorded, non-canceled trip in the list and
            // save it in the "tripidx" variable
            for (integer idx=0; idx < tripcount; idx++) {
              Trip trip = triplist[idx];
              if (!trip.trpcanceled) {
                tripidx = idx;
                break;
              }
            }
            // Check if a non-canceled trip has been found in the list of
            // recorded trips
            if (tripidx != null) {
              // Populate the "trlnewtrip" property of the trip log with the last
              // recorded, non-canceled trip in order to display it to the user
              cooobj.trlnewtrip = triplist[idx];
              // Invoke the confirmation dialog
              ->DialogCancelTrip;
            }
            else {
              // Throw an error if no non-canceled trip could be found
              throw coort.SetError(
                #ErrNoTripsToCancel,
                #ErrNoTripsToCancel.Print(null, cooobj.objname));
            }
          }
          else {
            // Throw an error if the trip log is not in "open" state
            throw coort.SetError(
              #ErrTripLogClosed,
              #ErrTripLogClosed.Print(null, cooobj.objname));
          }
        }
        // Display the dialog in read-only mode
        dialog DialogCancelTrip readonly {
          form = PageCancelTrip;
          target = cooobj;
          description = {}
          cancelbranch;
          nextbranch {
            caption = StrCancelTrip;
            symbol = SymbolWastebasket;
            expression {
              // Set the "Canceled" flag for the displayed trip and overwrite the
              // "trltrips" property of the trip log with the "triplist" variable
              triplist[tripidx].trpcanceled = true;
              cooobj.trltrips = triplist;
            }
          }
        }
        targetwindow = TARGETWINDOW_OVERLAY;
      }
    }
  }
}
```

wizards.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
```

```
import COOATTREDIT@1.1;
import COODESK@1.1;
import COOSEARCH@1.1;
import FSCVAPP@1.1001;
…
formpage PageCancelTrip {
  dataset {
    trlnewtrip;
  }
  layout {
    // Auto-generated layout block
    row {
      FSCLOGBOOK@111.100:trlnewtrip {
        detail = layout {
          row {
            FSCLOGBOOK@111.100:trpdepartureat {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpstartmileage {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trparrivalat {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpendmileage {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpduration {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpmileage {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpdepartureplace {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trpdrivername {
              colspan = 2;
              labelposition = left;
            }
          }
          row {
            FSCLOGBOOK@111.100:trpdestinationplace {
              colspan = 2;
              labelposition = left;
            }
            FSCLOGBOOK@111.100:trptype {
              colspan = 2;
              labelposition = left;
            }
          }
```

```
            row {
              FSCLOGBOOK@111.100:trppurpose {
                colspan = 4;
                labelposition = left;
              }
            }
          }
          rowspan = 2;
          colspan = 4;
          labelposition = top;
        }
      }
      row {
      }
    }
}
…
extend menu MenuCancelTripWizard {
  // Include the "long text" for the menu item, which is displayed in the tips pane,
  // in the "mlnames.lang" files
  menulongtext = {}
}
extend class TripLog {
  …
  menus {
    default = {
      expansions {
        MenuContextExpansion {
          condition = expression {
            cooobj.trlstate == TLS_OPEN;
          }
          entries = {
            MenuRecordTripWizard
          }
        }
        // Add the "Cancel Trip" menu item to the context menu by adding it to the
        // expansion point
        MenuContextExpansion {
          condition = expression {
            // Display the "Cancel Trip" menu item only if the trip log is in "open"
            // state and if it contains non-canceled trips
            cooobj.trlstate == TLS_OPEN &&
              count(cooobj.trltrips[!trpcanceled]) > 0;
          }
          entries = {
            MenuCancelTripWizard
          }
        }
        TaskPaneSelectedExpansion {
          condition = expression {
            cooobj.trlstate == TLS_OPEN;
          }
          entries = {
            MenuRecordTripWizard
          }
        }
        // Add the "Cancel Trip" menu item to the tips pane by adding it to the
        // expansion point
        TaskPaneSelectedExpansion {
          condition = expression {
            // Display the "Cancel Trip" menu item only if the trip log is in "open"
            // state and if it contains non-canceled trips
            cooobj.trlstate == TLS_OPEN &&
              count(cooobj.trltrips[!trpcanceled]) > 0;
```

```
            }
            entries = {
              MenuCancelTripWizard
            }
          }
        }
      }
    }
  }
}
```

strings.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  string StrCancelTrip;
}
```

errors.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  errormsg ErrNoTripsToCancel;
}
```

mlnames.lang

```
…
DialogCancelTrip.description            If you want to cancel the displayed trip click
                                        "<~#FSCLOGBOOK@111.100:StrCancelTrip.Print()~>"
                                        to continue or click "Cancel" to abort the
                                        operation.
DialogCancelTrip.mlname                 Cancel Trip
ErrNoTripsToCancel.errtext              There are no trips to cancel in trip
                                        log "%1$s".
MenuCancelTripWizard.menulongtext       Cancel the last recorded trip in the selected
                                        trip logs
MenuCancelTripWizard.menustatetext      Cancels the last recorded trip in selected trip
                                        logs
MenuCancelTripWizard.mlname             Cancel Trip
…
```

Figure 43 shows how to use the form designer to rename a property on a form page. For further details on how to use the form designer refer to [Faba11a].

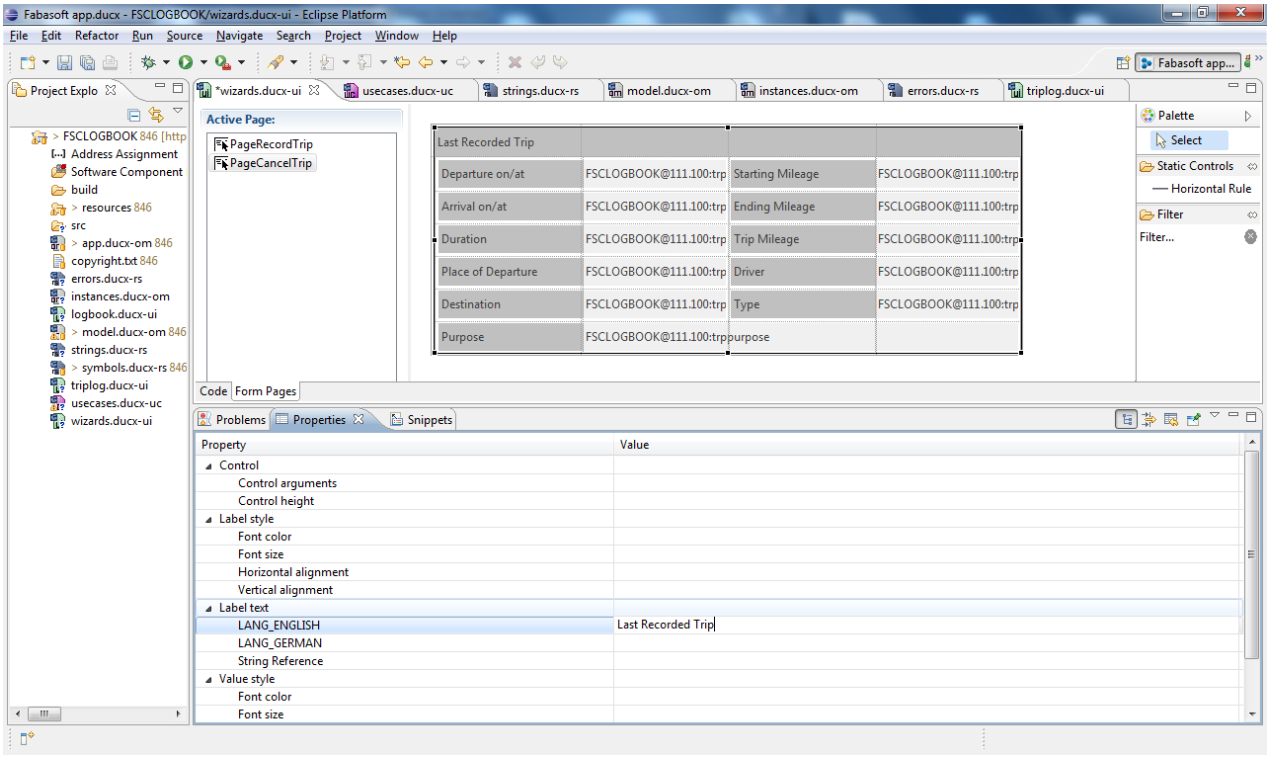


Figure 43: Renaming a property in the form designer

All done! You now have a fully working wizard for canceling recorded trips. Piece of cake, wasn't it?

6.7.3 Adding a wizard for closing a trip log

The last virtual application we're going to implement is the wizard for closing a trip log so that users can no longer record any trips in it.

So what's the point of this feature? Well, we don't want users to be able to make changes to a trip log after a certain event has occurred (e.g. sending a trip log to the department head for approval).

The wizard, depicted in Figure 44, follows the same principle as the wizards for recording and canceling trips, so implementing it won't be a challenge for you.

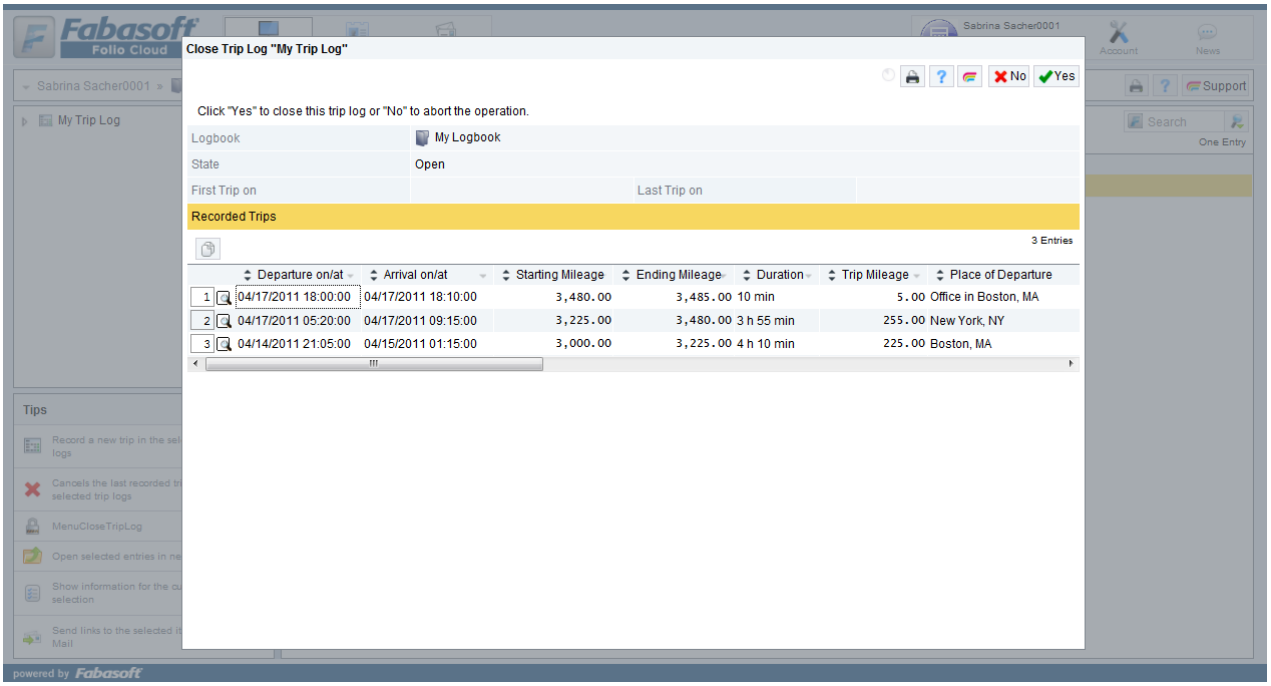


Figure 44: Wizard for closing a trip log

As this is mainly a repetition of the concepts already covered before, we've omitted most of the comments in the example code.

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  …
  menu usecase CloseTripLogWizard on selected or container {
    symbol = SymbolLock;
    variant TripLog {
      impl = application {
        expression {
          if (cooobj.trlstate == TLS_OPEN) {
            cooobj.ObjectLock(true, true);
            ->DialogCloseTripLog;
          }
          else {
            throw coort.SetError(#ErrTripLogClosed,
              #ErrTripLogClosed.Print(null, cooobj.objname));
          }
        }
        dialog DialogCloseTripLog readonly {
          // Display the same form page that is also used for reading/editing the
          // properties of a trip log
          form = PageTripLog;
          target = cooobj;
          description = {}
          cancelbranch {
            caption = StrNo;
            symbol = SymbolNo;
```

```
          }
          nextbranch {
            caption = StrYes;
            symbol = SymbolYes;
            expression {
              // Set the state of the trip log to "closed"
              cooobj.trlstate = TLS_CLOSED;
            }
          }
        }
        targetwindow = TARGETWINDOW_OVERLAY;
      }
    }
  }
}
```

wizards.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  …
  extend menu MenuCloseTripLogWizard {
    menulongtext = {}
  }
  extend class TripLog {
    …
    menus {
      default = {
        expansions {
          …
          MenuContextExpansion {
            condition = expression {
              cooobj.trlstate == TLS_OPEN;
            }
            entries = {
              MenuCloseTripLogWizard
            }
          }
          TaskPaneSelectedExpansion {
            condition = expression {
              cooobj.trlstate == TLS_OPEN;
            }
            entries = {
              MenuCloseTripLogWizard
            }
          }
        }
      }
    }
  }
}
```

mlnames.lang

```
…
DialogCloseTripLog.description          Click "Yes" to close this trip log or "No" to
                                        abort the operation.
DialogCloseTripLog.mlname               Close Trip Log "<~cooobj.objname~>"
MenuCloseTripLogWizard.menulongtext     Close the selected trip logs
MenuCloseTripLogWizard.menustatetext    Closes the selected trip logs
MenuCloseTripLogWizard.mlname           Close Trip Log
…
```

6.7.4 Adding a display action to show the number of recorded trips

In this chapter, we will show you how to implement a display action for the `trltrips` property of a trip log. The purpose of the display action is to render a string for the `trltrips` property that is shown when the property is displayed as a column of a list.

By default, when the `trltrips` property is displayed as a column of a list, the name of the property ("Recorded Trips") and the number of entries in the list of recorded trips (in square brackets) is shown (e.g. "Recorded Trips, ... [3]" if there are three recorded trips).

As this default display string is not all that appealing, we will improve the user experience by defining our own display action, which will display "1 trip" instead of "Recorded Trips, ... [1]" (in case of a single entry in the list of recorded trips) or "3 trips" instead of "Recorded Trips, ... [3]" (in case of three recorded trips) as illustrated by Figure 45.

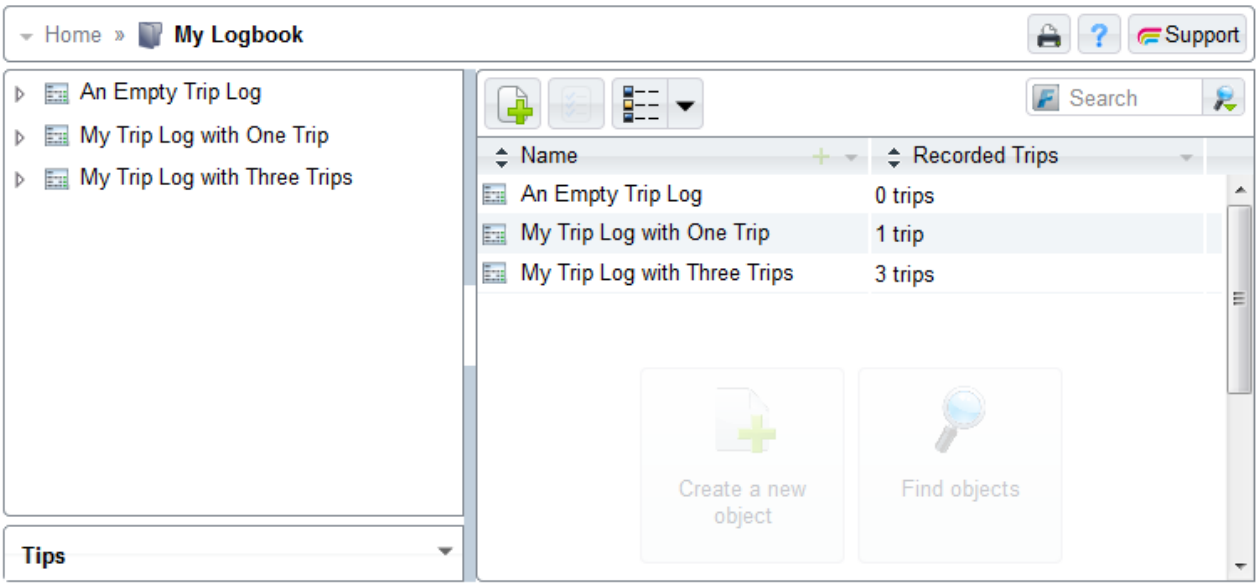


Figure 45: Displaying a property as a list column

Note: Refer to chapter "Defining the columns for the 'logtriplogs' property" on page 75 to learn how to define the default column settings for the lists of your Cloud App's object classes.

The following example documents all the necessary steps for the implementation of the display action. In addition to the display action, we also need to define two string objects for the multilingual string literals. Furthermore, we use the `display` keyword to assign the display action to the `trltrips` property.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  …
  class TripLog {
    …
    Trip[] trltrips readonly(ui) {
      // Assign the "AttrTripDisplay" display action to the "trltrips" property
      display = AttrTripDisplay;
    }
    …
  }
  …
}
```

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  …
  AttrTripDisplay(parameters as AttrGetDispPrototype) {
    variant TripLog {
      impl = expression {
        // Retrieve the number of entries in the list of recorded trips and filter out
        // the canceled entries
        integer tripcount = count(cooobj.trltrips[!trpcanceled]);
        // Use the COOSYSTEM@1.1:Print action to format the number of trips into the
        // appropriate string object holding the correct multilingual string and
        // return the formatted string in the output parameter named "string"
        string = (tripcount == 1)?
          #StrDisplayFormatSingleTrip.Print(null, tripcount) :
          #StrDisplayFormatMultipleTrips.Print(null, tripcount);
      }
    }
  }
}
```

strings.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  string StrDisplayFormatMultipleTrips;
  string StrDisplayFormatSingleTrip;
}
```

mlnames.lang

```
…
StrDisplayFormatMultipleTrips.string    %1$d trips
StrDisplayFormatSingleTrip.string       %1$d trip
…
```

6.7.5 Calculating the date of the first and last entry in a trip log

The next challenge we're going to tackle is the updating of the `trlfrom` and `trluntil` properties when the list of recorded trips is changed.

The `trlfrom` property must be populated with the departure date of the oldest recorded, non-canceled trip in the `trltrips` property and the `trluntil` property must contain the arrival date of the most recently recorded, non-canceled trip.

To accomplish this task, we define and implement a set action named `AttrTripsSet` and attach it to the `trltrips` property, as illustrated by the following example.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  …
  class TripLog {
    …
```

```
    Trip[] trltrips readonly(ui) {
      // Assign the "AttrTripsSet" set action to the "trltrips" property
      set = AttrTripsSet;
      display = AttrTripDisplay;
    }
    …
  }
  …
}
```

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  …
  AttrTripsSet(parameters as AttrSetPrototype) {
    variant TripLog {
      impl = expression {
        // The "value" parameter contains the list of recorded trips
        Trip[] triplist = value;
        if (triplist != null) {
          // Get only the non-canceled trips from the list
          triplist = triplist[!trpcanceled];
          if (triplist != null) {
            // Get the departure date from the oldest recorded trip and save it in the
            // "trlfrom" property
            cooobj.trlfrom = triplist[count(triplist)-1].trpdepartureat;
            // Get the arrival date from the most recently recorded trip and save it in
            // the "trluntil" property
            cooobj.trluntil = triplist[0].trparrivalat;
          }
        }
      }
    }
  }
}
```

6.8 Embedding Google visualizations

Let's add some color to your Cloud App by integrating a chart!

Fabasoft Folio Cloud makes it easy to integrate fancy charts in your Cloud App using the Google Chart Tools.

For a complete list of the available types of charts and additional documentation refer to `http://code.google.com/apis/visualization/documentation/gallery.html`.

To integrate a Google chart, you have to carry out the following steps:

- Add a reference to the `FSCGOOGLEVISUALS@1.1001` software component to your Cloud App project.
- Define an instance of `FSCGOOGLEVISUALS@1.1001:Visual` in an object model file and assign the desired chart type (represented by an instance of `FSCGOOGLEVISUALS@1.1001:Package`, e.g. `BarChart`, `ColumnChart`, `AreaChart`, `AnnotatedTimeLine`) using the `package` keyword. The so-called "visual" represents the actual chart and can be customized using the `params` keyword with the parameters described in the Google Chart Tools documentation (see `http://code.google.com/apis/visualization/documentation/gallery.html`).
- Define a data structure that corresponds to the data required by the chart you want to integrate. For instance, a bar chart requires different data than an annotated time line or a gauge chart.

- Define a property for providing the data to be displayed in the chart and assign it to an object class.
- Implement a get action for calculating the data and assign it to the property.
- Include the property on a form page and assign your visual to the property.

For our example, we will integrate an annotated time line chart showing the mileage information for every recorded trip of a trip log (see Figure 46).

So here's the detailed list of steps needed to integrate the chart depicted in Figure 46 into your Cloud App:

- Add a reference to the `FSCGOOGLEVISUALS@1.1001` software component to your Cloud App project.
- Define an instance of `FSCGOOGLEVISUALS@1.1001:Visual` with the reference `VisualTimeLine` in the `instances.ducx-om` file and assign it the `AnnotatedTimeLine` package using use the `package` keyword. Using the `params` keyword, you can customize the chart (e.g. define the thickness of the lines).
- Using the Google Chart Tools documentation, we can find out that for an annotated time line chart the first column is of type `datetime`, and specifies the X value on the chart and the Y value is a number that describes the corresponding date and time.
- In the `model.ducx-om` file, define a data structure with the reference `TimeLine` that corresponds to the requirements obtained from the Google Chart Tools documentation and is comprised of a `datetime` property with the reference `tltripdate` and a `float` property with the reference `tltripmileage`.
- As our chart should be part of a trip log, we add a property of type `TimeLine[]` to the `TripLog` object class and assign it the reference `trltimeline`. The property should only be calculated on demand and no values should be stored in it. Therefore, we add the `readonly` property modifier suffix (see [Faba11a]).
- In the `usecases.ducx-uc` file, we define and implement a get action with the reference `AttrTimeLineGet` for calculating the data for the chart. Afterwards, we assign the `AttrTimeLineGet` get action to the `trltimeline` property in the `model.ducx-om` file using the `get` keyword.
- Finally, we define a new form page with the reference `PageTimeLine` as part of the `FormTripLog` form in the `triplog.ducx-ui` file. In the `dataset` block, reference the `trltimeline` property and use the form designer to place the `trltimeline` property on the form page. Then assign the `FSCGOOGLEVISUALS@1.1001:ControlStdVisual` control to the `trltimeline` property by selecting the property on the form designer canvas, right-clicking it to open the context menu and selecting the `FSCGOOGLEVISUALS@1.1001:ControlStdVisual` control. After that, select the "Properties" view and enter `FSCLOGBOOK@111.100:VisualTimeLine` in the *Control arguments* field (see Figure 47).

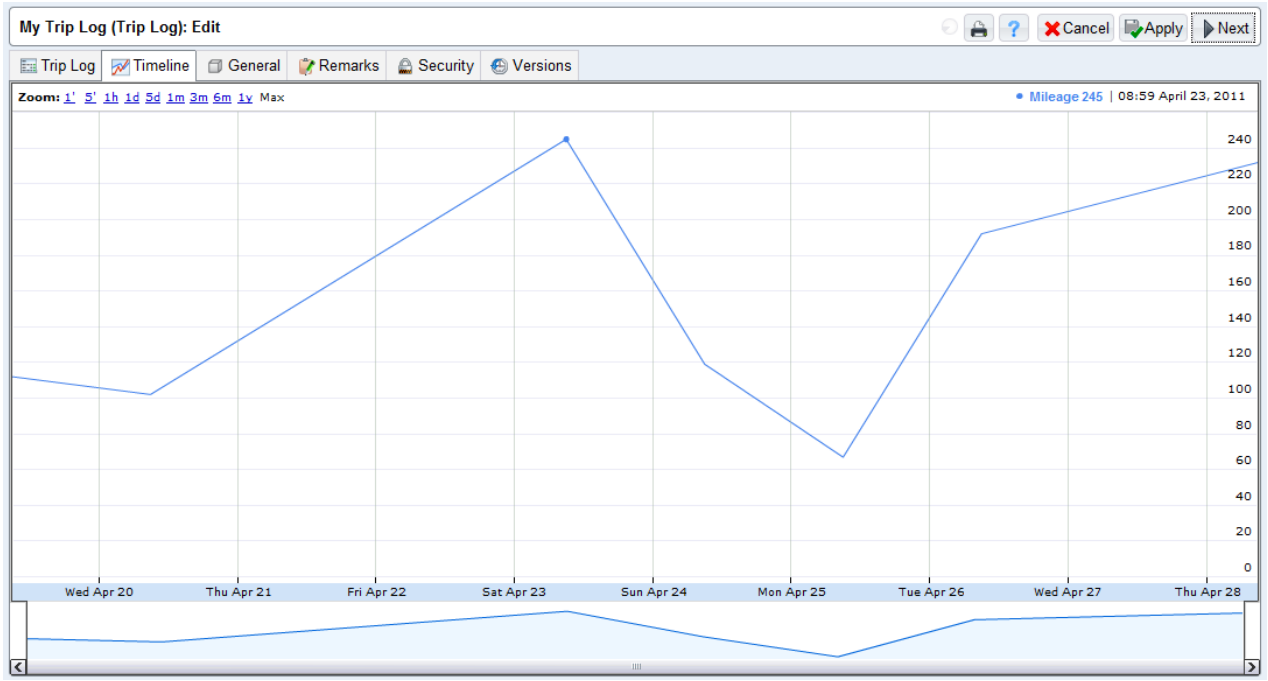


Figure 46: Using a Google Chart to display mileage information

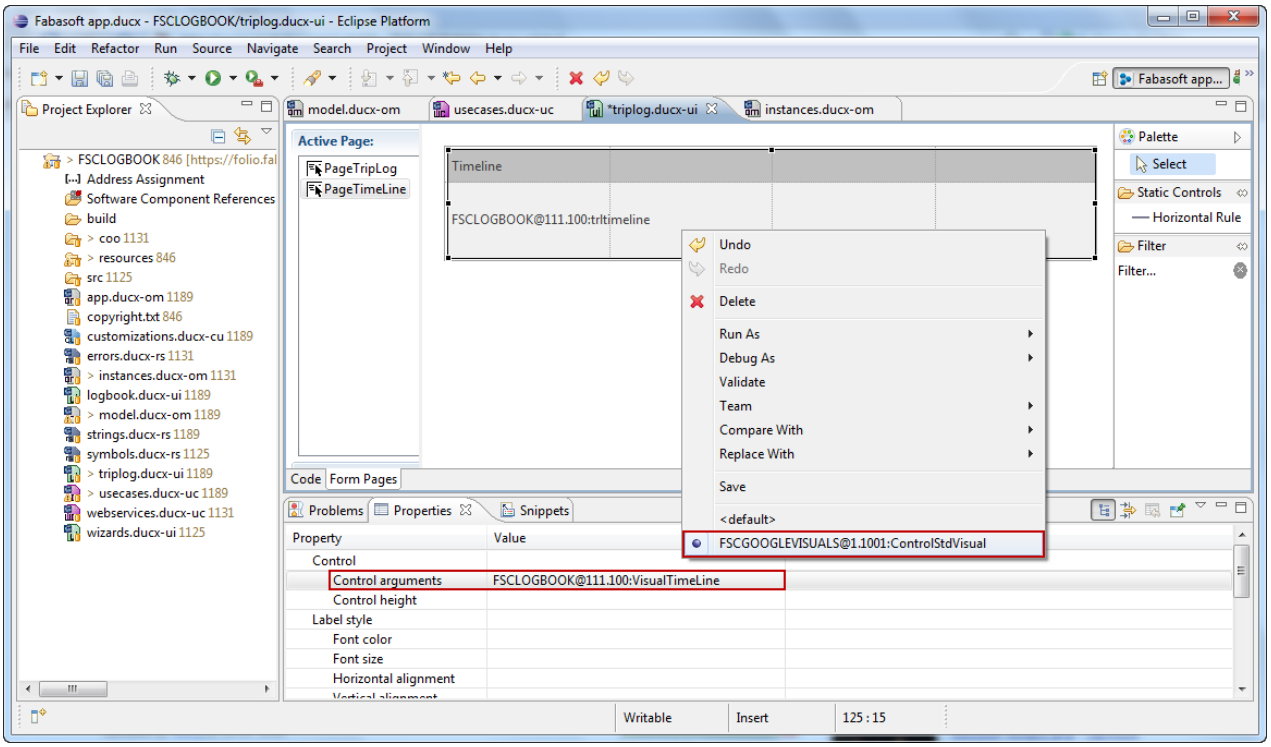


Figure 47: Assigning a control to a property

The following example documents all the required steps to integrate the annotated time line chart in your Cloud App.

Example

instances.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCGOOGLEVISUALS@1.1001;
  …
  instance Visual VisualTimeLine {
    package = AnnotatedTimeLine;
    params<key, value> = {
      // Increase the thickness of the lines in the chart
      {"thickness", "3"}
    }
  }
}
```

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  …
  struct TimeLine {
    datetime tltripdate;
    unsigned float(6,2) tltripmileage;
  }
  class TripLog {
    …
    TimeLine[] trltimeline readonly {
      get = AttrTimeLineGet;
      copy = NoOperation;
    }
  }
  …
}
```

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  …
  AttrTimeLineGet(parameters as AttrGetPrototype) {
    variant TripLog {
      impl = expression {
        // Define a variable for building the time line data
        TimeLine[] timeline;
        // Iterate over all the non-canceled recorded trips
        for (Trip trip : cooobj.trltrips[!trpcanceled]) {
          // Add an entry to the "timeline" list and populate it with the date and
          // time of departure and the trip mileage of the current trip
          timeline += {
            tltripdate = :>trip.trpdepartureat,
            tltripmileage = :>trip.trpmileage
          };
        }
        // Return the time line data in the "value" variable
        value = timeline;
      }
    }
  }
```

```
}
```

triplog.ducx-ui

```
userinterface FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  form FormTripLog {
    formpage PageTripLog {
      …
    }
    formpage PageTimeLine {
      symbol = SymbolChart;
      dataset {
        trltimeline;
      }
      layout {
        // Auto-generated layout block
        row {
          FSCGOOGLEVISUALS@1.1001:ControlStdVisual("FSCLOGBOOK@111.100:VisualTimeLine")
            FSCLOGBOOK@111.100:trltimeline {
            rowspan = 2;
            colspan = 4;
            labelposition = top;
          }
        }
        row {
        }
      }
    }
  }
  …
}
```

In addition to the Google Chart Tools, Fabasoft Folio Cloud also supports the jQuery JavaScript library, which allows you to build and integrate sophisticated custom controls for creating fantastic user experiences such as management dashboards.

Elaborating on the jQuery JavaScript library goes beyond the scope of this book. For further information on the jQuery JavaScript library refer to `http://www.jquery.com`.

6.9 The finishing touches

So far, so good! We've come a long way already, but there are still a few flaws in your Cloud App that we need to fix in order to make it perfect.

In this chapter, we'll give your Cloud App the finishing touches before we start with the testing phase.

6.9.1 Defining a name build for the 'TripLog' object class

In many situations, you want the name of the instances of a particular object class to be built automatically based on rules so that users don't have to manually assign a name.

For our trip logs, we will define a name build that automatically generates a name for each instance that is comprised of the values of the `trllogbook`, `trlfrom`, `trluntil` and `trlstate` properties.

A generated name for a trip log should look like this example:

```
Open Trip Log for "My Logbook" (May 1, 2011 – Apr 30, 2012)
```

To implement this requirement, we have to carry out the following steps:

- Define a `NameBuild` customization for building the name of a trip log
- Override the `FSCCONFIG@1.1001:EvaluateGenericNameBuild` action with the `FSCCONFIG@1.1001:MethodGenericNameBuild` method definition in object class `TripLog`
- Define pattern strings for formatting the name of a trip log

Note: All customizations must be defined in a Fabasoft app.ducx customization file with a `.ducx-cu` extension. For further information on customization points and customizations refer to [Faba11a].

The following example demonstrates how to implement the name build for trip logs described before.

In order to suppress the default dialog box that is displayed when creating a new object (i.e. for entering a name), we also override the `COODESK@1.1:RenameObject` action with the virtual application `FSCVAPP@1.1001:NoOperation`.

Example

customizations.ducx-cu

```
customization FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCCONFIG@1.1001;
  default {
    customize NameBuild<TripLog> {
      // Reference all properties that have an impact on the generated name of the
      // object in the "properties" block
      properties = {
        trlfrom,
        trluntil,
        trllogbook,
        trlstate
      }
      // The "build" expression must yield a string that is set to the new name of
      // the trip log
      build = expression {
        // Get the date of the first and last recorded trip from the trip log
        datetime from = cooobj.trlfrom;
        datetime until = cooobj.trluntil;
        if (from != null && until != null) {
          // If the trip log already contains recorded trips, the name should be
          // generated according to the pattern defined in the
          // "StrTripLogNameFormatLong" string
          return #StrTripLogNameFormatLong.Print(null, cooobj.Format(cooobj.trlstate),
            cooobj.trllogbook.objname, cooobj.Format(from, "d"),
            cooobj.Format(until, "d"));
        }
        else {
          // If the trip log does not contain any recorded trips yet, the name should
          // be generated according to the pattern defined in the
          // "StrTripLogNameFormatShort" string
          return #StrTripLogNameFormatShort.Print(null, cooobj.Format(cooobj.trlstate),
            cooobj.trllogbook.objname);
        }
      }
    }
  }
}
```

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
```

```
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  import FSCCONFIG@1.1001;
  …
  override EvaluateGenericNameBuild {
    variant TripLog {
      impl = MethodGenericNameBuild;
    }
  }
  override RenameObject {
    variant TripLog {
      impl = FSCVAPP@1.1001:NoOperation;
    }
  }
}
```

strings.ducx-rs

```
resources FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  string StrTripLogNameFormatShort;
  string StrTripLogNameFormatLong;
}
```

mlnames.lang

```
…
StrTripLogNameFormatLong.string    %1$s Trip Log for "%2$s" (%3$s - %4$s)
StrTripLogNameFormatShort.string   %1$s Trip Log for "%2$s"
…
```

6.9.2 Establishing an ACL reference between trip log and logbook

The access control list (ACL) of an object governs which users or teams get access to the object as well as the level of access they get (e.g. read access, change access or full control).

By default, objects created outside of a team room are assigned the ACL “ACL for Objects Without a Team Room” whereas objects created within a team room get the ACL “ACL for Objects in Team Rooms”.

For further information on ACLs refer to [Faba11a] and the Fabasoft Folio Cloud online help.

In your Cloud App, trip logs always belong to a logbook and should also share the same ACL.

Instead of having to worry about keeping the ACLs of a logbook and the associated trip logs in sync, we use a better mechanism for making sure that trip logs always get the same ACL as the logbook they belong to: We establish an ACL reference that points from the trip log to the logbook. Once the ACL reference has been established, the ACL of the parent (i.e. the logbook) is also used for the subordinated objects (i.e. the trip logs).

To implement this requirement, create a new Fabasoft app.ducx use case file named `overrides.ducx-uc` and add an override for the `COOSYSTEM@1.1:ObjectConstructor` action as depicted in the following example.

Example

overrides.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  import COODESK@1.1;
```

```
  override ObjectConstructor {
    variant TripLog {
      impl = expression {
        // Invoke the super method
        super();
        // Get the container object where the trip log was created in
        Object containerobj = cooobj.GetSelected()[8];
        // Check if the container object is a logbook
        if (containerobj != null && containerobj.HasClass(#Logbook)) {
          // Set an ACL reference to the logbook and remove the pointer to the
          // current ACL of the trip log
          cooobj.objaclref = containerobj;
          cooobj.objaclobj = null;
        }
      }
    }
  }
}
```

Refer to [Faba11a] for detailed information about triggers in Fabasoft Folio Cloud.

6.9.3 Deleting the trip logs along with the logbook

When deleting a logbook, the associated trip logs should be automatically deleted along with the logbook. To accomplish this, assign the `COOSYSTEM@1.1:AttrChildrenDestructor` destructor action to the `logtriplogs` property using the `dtor` keyword.

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  …
  class Logbook : CompoundObject {
    …
    unique TripLog[] logtriplogs {
      link = trllogbook;
      child = true;
      // Destructor action for deleting the trip logs along with the logbook
      dtor = AttrChildrenDestructor;
    }
  }
}
```

6.9.4 Things to consider when dealing with team rooms

Team rooms are the central instrument for collaboration in Fabasoft Folio Cloud. Therefore, your Cloud App must be able to support team rooms. More precisely, it must be possible to create or move the instances of your object classes into a team room without breaking your Cloud App's functionality.

There are a few things that you have to be aware of when dealing with team rooms:

- By default, all objects of a team room are assigned the `FSCFOLIOCLOUD@1.1001:TeamRoomObjectsACL` ACL, which grants access to all members of the team room.
- When an object is created within or moved into a team room, it becomes part of that team room, it is assigned the `FSCFOLIOCLOUD@1.1001:TeamRoomObjectsACL` ACL, its owner is changed to the owner of the team room and the object counts towards the team room owner's quota.
- The object's children also should become part of the team room.

When an object is assigned to a team room, the objects referenced in an object pointer or object list property are not automatically assigned to the parent object's team room unless the `child` keyword has been set to `true` for the corresponding property.

Let us explain that in the context of our Cloud App:

- A logbook is comprised of trip logs that logically belong to the logbook.
- If you move a logbook into a team room, it becomes part of the team room. The owner of the logbook object is changed to the team room's owner, and its ACL is set to `FSCFOLIOCLOUD@1.1001:TeamRoomObjectsACL`.
- The trip logs of the logbook are not assigned to this team room unless you set the `child` property to `true` for the `logtriplogs` property. That's bad, because they logically belong to the logbook and therefore should also be part of the same team room as the logbook.

If child objects are not assigned to the same team room as their parent, a bunch of problems may arise:

- They are not automatically assigned the `FSCFOLIOCLOUD@1.1001:TeamRoomObjectsACL` ACL, which means that the team room members may be unable to access them.
- The owner is not changed to the team room's owner, so they count towards the quota of their original owner.

It basically all boils down to setting the `child` property to `true` for all of your object pointer properties and lists referencing objects that logically are children of the current object.

In your Cloud App, there is only one parent-child relationship so far. Therefore, we only had to set the `child` keyword to `true` for the `logtriplogs` property, which is pointing from the logbook to the associated trip logs.

Note: You don't have to set the `child` keyword to `true` for the `trllogbook` property of the trip log (which is the one pointing from the trip log back to the logbook it belongs to), because it only needs to be set on the property belonging to the parent object (which, in our example, is the logbook).

The following example shows how to set the `child` keyword to `true` for an object pointer or object list property. For further information refer to [Faba11a].

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  …
  class Logbook : CompoundObject {
    …
    unique TripLog[] logtriplogs {
      …
      child = true;
      …
    }
  }
}
```

6.9.5 Activating the license check for your object classes

If a user does not have a valid license for your Cloud App, all of the menu entries (e.g. in the context menu and the tips pane) provided by your Cloud App are automatically disabled and the user cannot create any instances of the object classes of your Cloud App.

However, by default the user can still access existing instances of your object classes (created by another user with a valid license for your Cloud App), and read and edit the properties of these objects (provided that they have sufficient access rights).

For example, if Wanda Carney acquires a license for your Cloud App, creates a new logbook within a team room and then invites William Briere to the team, William will be able to read and edit the properties of Wanda's logbook. Nevertheless, William will not be able to create trips logs within the logbook, record or cancel trips.

If you want to prevent users without a valid license for your Cloud App from accessing objects belonging to your Cloud App (see Figure 48), you have to assign the app object representing your Cloud App (`AppFSCLOGBOOK`) to the `COOATTREDIT@1.1:compapps` property of the object classes of your Cloud App (see example).

Figure 48: Cloud App license check prompting a user to obtain a valid license

Example

model.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import FSCTERM@1.1001;
  …
  class TripLog {
    compapps = AppFSCLOGBOOK;
    …
  }
  class Logbook : CompoundObject {
    compapps = AppFSCLOGBOOK;
    …
  }
  …
}
```

6.9.6 Reacting to app state changes

You can define an app state change action that is invoked when a user activates or deactivates your Cloud App, e.g. to perform initialization or cleanup steps.

The `COOATTREDIT@1.1:AppStateChangeActionPrototype` prototype must be assigned to the app state change action.

Note: Your implementation of the app state change action must be error tolerant. For instance, you might have to deal with situations where you can't access the objects you want to process because they are locked or protected with a security level.

Example

app.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import COOATTREDIT@1.1;
  import COOTC@1.1001;
  instance App AppFSCLOGBOOK {
    …
    appstatechangeaction = ChangeAppState;
  }
}
```

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  import FSCCONFIG@1.1001;
  import COOATTREDIT@1.1;
  …
  ChangeAppState(parameters as AppStateChangeActionPrototype) {
    variant App {
      impl = expression {
        if (active != oldstate) {
          if (active) {
            // TODO: Cloud App has been activated, perform initializations...
          }
          else {
            // TODO: Cloud App has been deactivated, perform cleanup...
          }
        }
      }
    }
  }
}
```

6.9.7 Defining an app category

An app category allows you to combine the object classes belonging to your Cloud App into a distinct category that is displayed when a user is creating a new object.

Defining an app category for your Cloud App is a good approach for getting more user attention as it is prominently featured in the “Create” dialog box depicted in Figure 49.

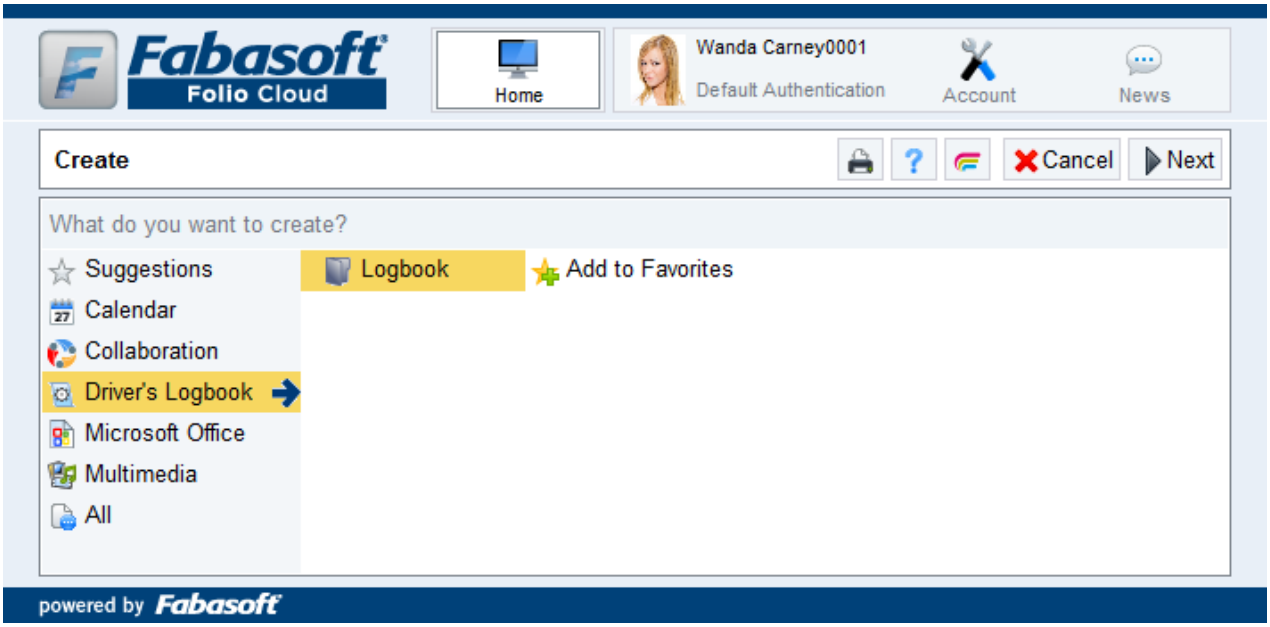


Figure 49: App categories in the “Create” dialog box

To define an app category, add a reference to the `COOTC@1.1001` software component and use the `instance` keyword to define an app category instance with the reference `AppCategoryLogbook` in the `app.ducx-om` file. Then add the object classes belonging to your Cloud App to the `templates` list of the app category.

Example

app.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import COOATTREDIT@1.1;
  import COOTC@1.1001;
  instance App AppFSCLOGBOOK {
    …
  }
  instance AppCategory AppCategoryLogbook {
    symbol = SymbolAppFSCLOGBOOK;
    apps = {
      AppFSCLOGBOOK
    }
    templates = {
      Logbook,
      TripLog
    }
  }
}
```

6.9.8 Defining the context-sensitive help

At the beginning of the object modeling section, we mentioned that you are required to document your source code using Javadoc style comments.

Unfortunately, that’s not all of the documentation work you have to do as a Cloud App developer.

At Fabasoft, we strongly believe that a solid Cloud App has to come with solid user documentation in order to deliver an exceptional user experience, and this is why we also require you to provide context-sensitive help for your Cloud App.

Users can easily turn the context-sensitive help on and off by a simple click on the “?” button available in all dialogs (which is visible unless it has been explicitly deactivated for a dialog). Figure 50 shows how the context-sensitive help is displayed in a typical dialog.

So how can you provide context-sensitive help for your Cloud App?

You may have already noticed the `explanations.userdoc` files underneath the language-specific folders in the `resources` folder of your Cloud App project. This is where you have to define the context-sensitive help.

Basically, an `explanations.userdoc` file is content in XML format comprising an `explanation` node for each element of your Cloud App that needs to be included in the context-sensitive help. The Fabasoft app.ducx compiler automatically updates the files when you add new properties and other elements to your Cloud App.

All you need to do is replace the “Your content goes here!” placeholders with the actual help text.

Be aware that you have to obey XML formatting and escaping rules when modifying the `explanations.userdoc` files.

Example

explanations.userdoc

```
<?xml version="1.0" encoding="UTF-8"?>
<explanations xmlns="http://www.fabasoft.com/ducx/explain/20090309#">
  <explanation reference="trllogbook" type="detail">
    <content>
      <b>Logbook</b>
      <p>This field displays the logbook this trip log belongs to.</p>
    </content>
  </explanation>
  <explanation reference="trlstate" type="detail">
    <content>
      <b>State</b>
      <p>This field displays the current state of the trip log.</p>
    </content>
  </explanation>
  <explanation reference="trluntil" type="detail">
    <content>
      <b>Last Trip on</b>
      <p>This field displays the arrival date of the last trip recorded in this trip.
      </p>
    </content>
  </explanation>
  <explanation reference="trlfrom" type="detail">
    <content>
      <b>First Trip on</b>
      <p>This field displays the departure date of the first trip recorded
         in this trip log. </p>
    </content>
  </explanation>
  <explanation reference="trltrips" type="detail">
    <content>
      <b>Recorded Trips</b>
      <p>This field displays the list of trips recorded in this trip log.</p>
    </content>
  </explanation>
  …
</explanations>
```

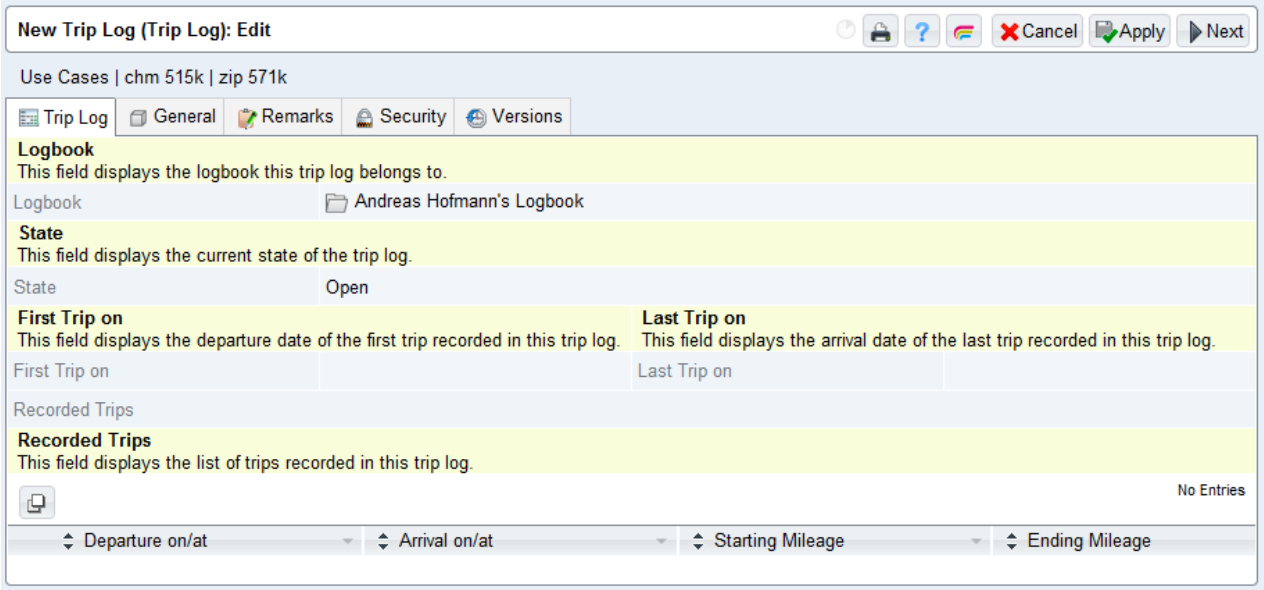


Figure 50: Displaying context-sensitive help

6.10 Advanced stuff

In this chapter, we will discuss some of the more advanced stuff, such as the integration of web services, and provide a brief overview of other APIs you can use in the context of your Cloud App.

6.10.1 Creating a web service

Creating an SOAP web service is easy! All you need to do is define and implement the web service methods, and then define a web service definition where you reference the web service methods that you want to expose as a web service.

Note: Any use case can be exposed as a web service method. You just need to make sure that it is implemented on object class `COOSYSTEM@1.1:User`. As a convention, all web service methods should be prefixed with `SOAP`.

In our example, we define three web service methods for creating, updating and deleting logbooks in a new Fabasoft app.ducx use case file named `webservices.ducx-uc`. For the sake of brevity, we will not elaborate on the actual implementation details.

We define a web service definition in the `instances.ducx-om` file, where we reference the use cases we want to expose as web methods of the web service. In the web service definition, we can also assign the external names of the web service methods (e.g. `CreateLogbook`, `UpdateLogbook` and `DeleteLogbook`).

Example

webservices.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  usecase SOAPCreateLogBook(string name, string vehicleid, retval Logbook logbook) {
    variant User {
      impl = expression {
```

```
        if (name != null && vehicleid != null) {
          Logbook logbook = #Logbook.ObjectCreate();
          logbook.objname = name;
          logbook.logvehicleid = vehicleid;
          logbook.ShareObject(null, null, #objchildren, cooroot);
        }
        else {
          throw #ErrMissingInformation;
        }
      }
    }
  }
  usecase SOAPUpdateLogBook(Logbook logbook, optional string name,
    optional string vehicleid) {
    variant User {
      impl = expression {
        if (logbook != null) {
          logbook.ObjectLock(true, true);
          if (logbook.objname != name) {
            logbook.objname = name;
          }
          if (logbook.logvehicleid != vehicleid) {
            logbook.logvehicleid = vehicleid;
          }
        }
        else {
          throw #ErrMissingInformation;
        }
      }
    }
  }
  usecase SOAPDeleteLogBook(Logbook logbook) {
    variant User {
      impl = expression {
        if (logbook != null) {
          logbook.ObjectDelete();
        }
        else {
          throw #ErrMissingInformation;
        }
      }
    }
  }
}
```

instances.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  import FSCOWS@1.1001;
  import FSCGOOGLEVISUALS@1.1001;
  …
  instance WebServiceDefinition WebService {
    webserviceactions<webserviceoperation, webserviceaction> = {
      {"CreateLogBook", SOAPCreateLogBook},
      {"UpdateLogBook", SOAPUpdateLogBook},
      {"DeleteLogBook", SOAPDeleteLogBook}
    }
  }
}
```

errors.ducx-rs

```
resources FSCLOGBOOK@111.100
{
```

```
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  …
  errormsg ErrMissingInformation;
}
```

mlnames.lang

```
…
ErrMissingInformation.errtext           You did not provide all the required data.
```

Using your Cloud App VDE, you can generate a web service description language (WSDL) document for web services and web methods.

A special URL allows you to retrieve the WSDL document for a web service definition in your Cloud App VDE:

`https://folio.fabasoft.com/dev<X>/vm<Y>/folio/fscdav/wsdl?WEBSVC=<web service definition>`

The `WEBSVC` URL parameter is used to reference the web service definition. You can either specify the fully qualified reference or the address of the web service definition. If you specify the fully qualified reference, you must replace special characters (`@`, colons and dots) with underscores.

The following example URL generates the WSDL document for our web service definition with the reference `FSCLOGBOOK@111.100:WebService`:

`https://folio.fabasoft.com/dev2/vm23/folio/fscdav/wsdl?WEBSVC=FSCLOGBOOK_111_100_WebService`

Figure 51 shows part of the WSDL document generated for our web service.

```
- <xs:element name="CreateLogBook">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="name" type="xs:string" maxOccurs="1" minOccurs="0"/>
        <xs:element name="vehicleid" type="xs:string" maxOccurs="1" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
- <xs:element name="CreateLogBookResponse">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="logbook" type="xs:string" maxOccurs="1" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
- <xs:element name="UpdateLogBook">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="logbook" type="xs:string" maxOccurs="1" minOccurs="0"/>
        <xs:element name="name" type="xs:string" maxOccurs="1" minOccurs="0"/>
        <xs:element name="vehicleid" type="xs:string" maxOccurs="1" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
- <xs:element name="UpdateLogBookResponse">
    <xs:complexType/>
  </xs:element>
```

Figure 51: WSDL output generated for the web service

Note: The URLs mentioned before may only be used for writing web service consumers that connect to your Cloud App VDE for testing purposes. Once your Cloud App has been deployed into the Fabasoft Folio Cloud production system, use the following URL to reach out to your web service definition from a web service consumer:

`https://folio.fabasoft.com/folio/fscdav/wsdl?WEBSVC=<web service definition>`

Refer to [Faba11f] for an introduction on how to create a web service consumer using either Eclipse, the Axis2 code generator and Java; Microsoft Visual Studio and C# or Eclipse; or Fabasoft app.ducx and expression language.

6.10.2 Other supported APIs

Fabasoft Folio Cloud supports a plethora of open standard APIs such as WebDAV, CalDAV (see [IETF07] and [Faba11g]) and CMIS (see [OASI10] and [Faba11h]) to make it as simple as possible for you to integrate or consume virtually any type of external resource within your Cloud App.

Note: The software component `FSCOWS@1.1001` contains useful actions for invoking SOAP web service methods (`CallSoapXML`, `CallSoapXMLEx`) as well as for sending generic HTTP requests (`SendHttpRequest`).

Elaborating on these technologies and APIs ventures beyond the scope of this book. For further information refer to the mentioned white papers and the resources listed in the chapter “Getting help, code samples and support” on page 151.

6.11 Tracing and debugging

Sometimes things just don’t work out at the very first shot. That’s why we’ve included some nifty features that allow you to add trace messages and debug through your code so that you can easily and efficiently track down any runtime issues that might arise during development.

6.11.1 Tracing in Fabasoft app.ducx expression language

The built-in tracing functionality of Fabasoft app.ducx allows you to produce extensive traces of your use case implementations.

The trace output contains detailed information about all use cases invoked by your use case implementations, along with the parameter values passed to and returned by the invoked use cases to allow you to get a complete picture of the call stack.

Using Fabasoft app.ducx expression language, you can use the `%%TRACE` directive to write custom trace messages to the Fabasoft app.ducx Tracer.

The `%%TRACE` directive can also be used to trace special objects like `cootx` to output all transaction variables defined for a transaction or `coometh` to output all set parameters within the implementation of a method.

Values traced using the `%%TRACE` directive are only written to the Fabasoft app.ducx Tracer if trace mode is enabled for your Cloud App in the project preferences. Your trace messages, therefore, don’t have a performance impact when tracing is deactivated.

To enable tracing, select your Cloud App project in Project Explorer, open the context menu and select “Properties”. Open the “Fabasoft app.ducx” page, select *Enable tracing* and click “OK” (see Figure 24 on page 49).

You can view the trace output of the Fabasoft app.ducx Tracer in the “Console” view of Eclipse. To activate the “Console” view select “Window” > “Show View” > “Other”, then select “Console” from the “General” branch and click “OK”.

Before you can see the trace output in the “Console” view, you have to start a tracing session. To start a tracing session, click the “Start Tracing Session” button in the button bar of the “Console” view (see Figure 52). While the tracing session is active, all trace output is retrieved from the Cloud App VDE and logged in the “Console” view of Eclipse. When you’re done, click the “Stop Tracing Session” button to stop the tracing session.

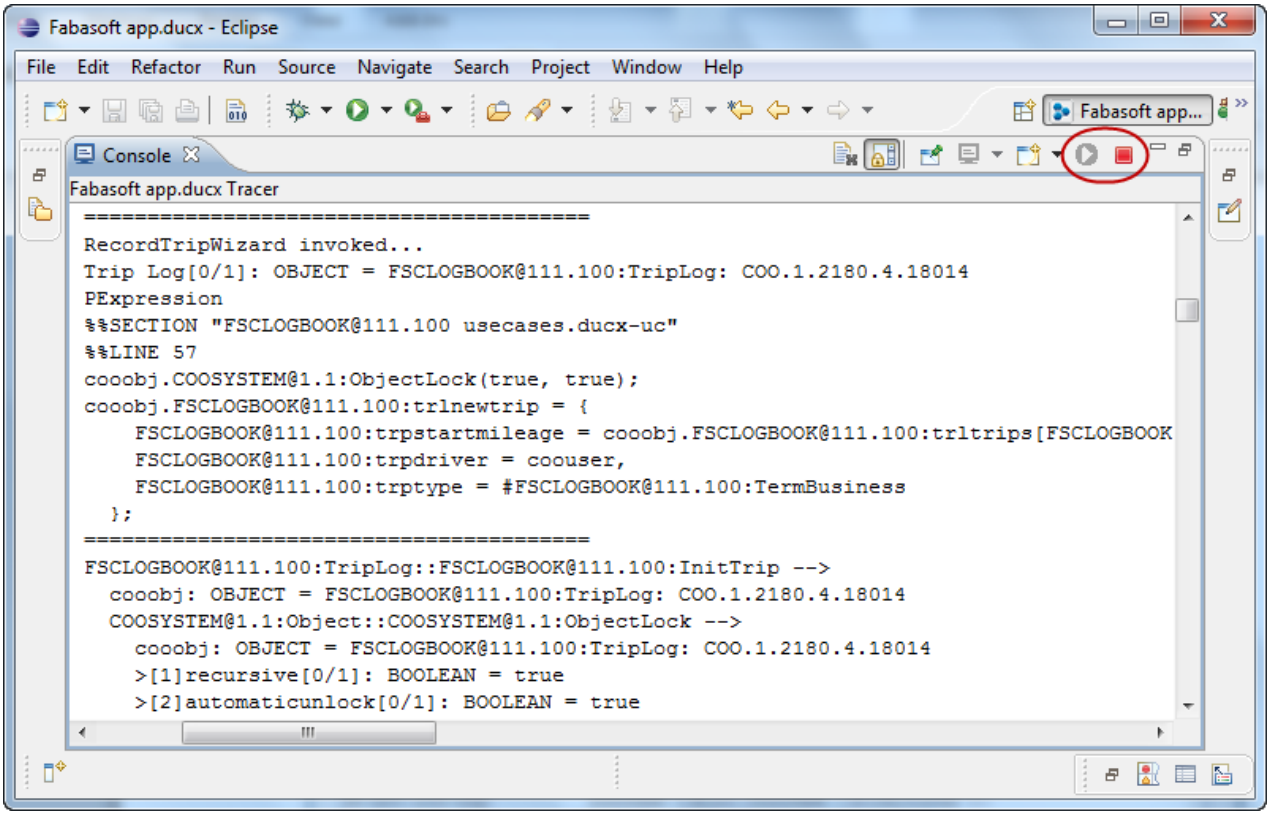


Figure 52: Viewing the trace output of the Fabasoft app.ducx Tracer in Eclipse

Alternatively, point your browser to the Cloud App VDE self-service portal (see chapter “Working with the Cloud App VDE” on page 35) and select “View Trace Output” to view the trace output on your Cloud App VDE.

You can also access the trace output using the following URL:

`https://folio.fabasoft.com/dev<X>/vm<Y>/selfservice/trace.php`

By default, the last 200 lines of trace output are displayed. Using the `lines` parameter you can define the number of lines of trace output you want to see. The maximum number of lines is 10,000 (e.g. `https://folio.fabasoft.com/dev2/vm23/selfservice/trace.php?lines=1000`).

The following example demonstrates how to use trace directives in Fabasoft app.ducx expression code, and the resulting trace output produced is depicted in Figure 53.

Example

usecases.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  menu usecase RecordTripWizard on selected or container {
    symbol = SymbolRecordTrip;
    variant TripLog {
      impl = application {
        expression {
          // Custom trace message
          %%TRACE "RecordTripWizard invoked...";
          // Write the contents of the "cooobj" variable to the tracer
```

```
            %%TRACE("Trip Log", cooobj);
            if (cooobj.trlstate == TLS_OPEN) {
              cooobj.InitTrip();
              ->DialogRecordTrip;
            }
            else {
              throw coort.SetError(
                #ErrTripLogClosed,
                #ErrTripLogClosed.Print(null, cooobj.objname));
            }
          }
          …
        }
      }
    }
}
```

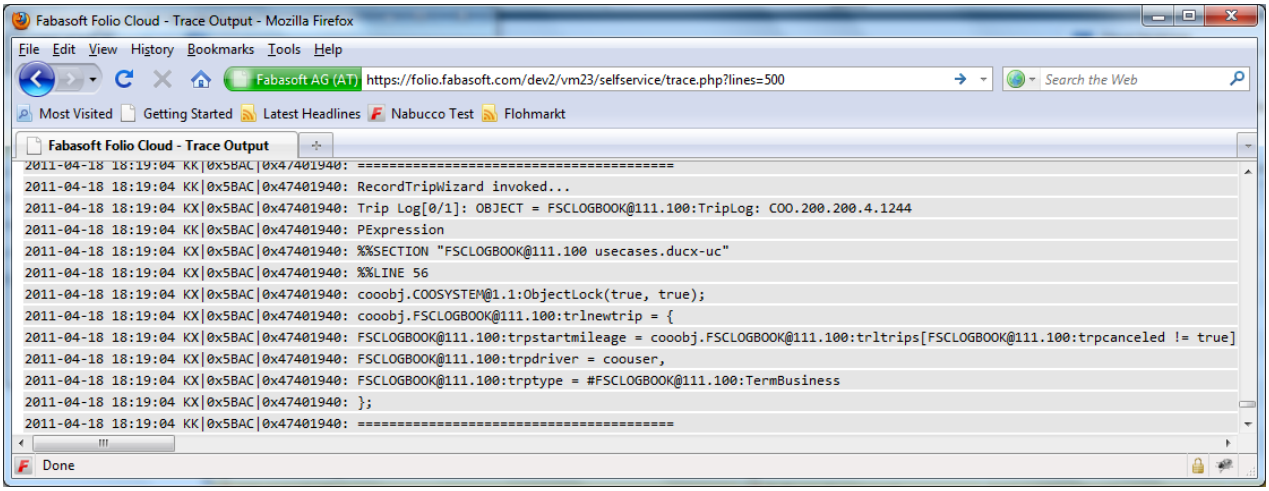


Figure 53: Viewing the trace output of the Fabasoft app.ducx Tracer in the self-service portal

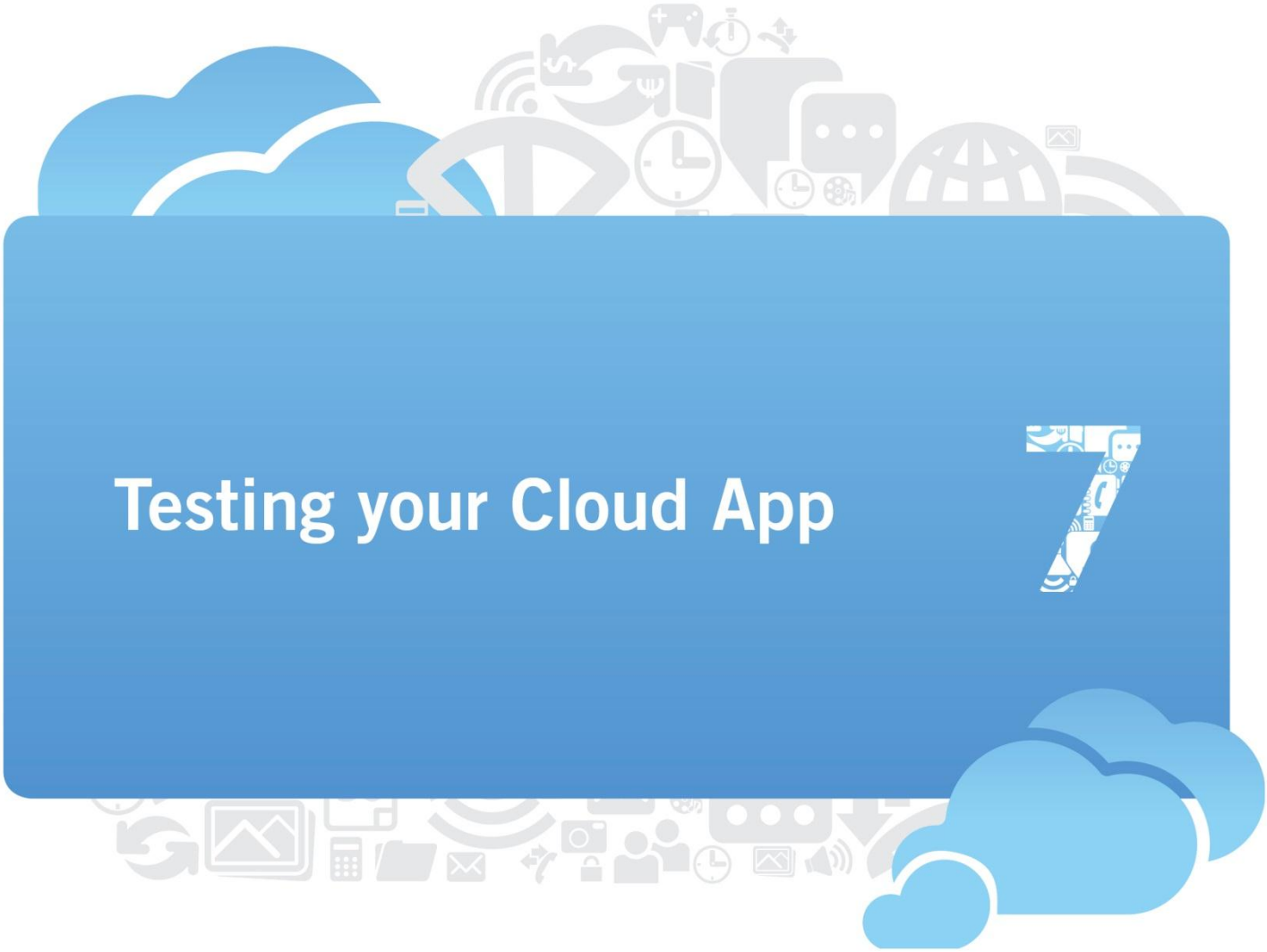
For detailed information on tracing refer to [Faba11a].

6.11.2 Debugging your Cloud App

The Fabasoft app.ducx debugger allows you to include breakpoints in your code and step through your use case implementations from within the Eclipse IDE.

You can set a breakpoint by double clicking in the column just in front of your app.ducx source code or by using the `%%debugger` directive in your expression code.

Refer to [Faba11a] to learn how to configure and activate the Fabasoft app.ducx debugger.



Testing your Cloud App

7

7 Testing your Cloud App

We all know that testing is important. Today's test-driven development methodologies even require us to write our tests as we go rather than at the end of the process.

Still, for most of us developer folks coding is much more fun, so we usually tend to focus on the part that's more fun and procrastinate when it comes to testing.

Well, we've got bad news for you: If you want to see your Cloud App going live anytime soon you have to embrace the idea of writing tests whenever you add or change something in your code.

Testing takes time. Don't forget to allocate a fair amount of time to designing, writing and updating robust tests also taking into account the corner cases. It's not in vain though, because in Fabasoft Folio Cloud quality pays dividends.

Why are we so strict when it comes to testing?

Because creating and continuously running a battery of automated tests is the only way you can keep huge amounts of existing code from breaking as you integrate new code.

Every Cloud App contributes new code into Fabasoft Folio Cloud, and the only way to maintain our standard of exceptional code quality is to enforce to our processes that uphold quality above all.

So what does this mean for you as a Cloud App developer?

Basically, it means that you have to produce the following deliverables in addition to your actual Cloud App:

- Unit tests for all of your use cases that are not implemented as virtual applications
- Fabasoft app.test tests for all of your Fabasoft app.ducx expression code that cannot be covered by unit tests
- A combined code coverage ratio of 100 %

7.1 Creating and running unit tests

Unit tests are an integral and essential part of every modern software development methodology and provide a quick way to boost your code coverage. In this chapter you will learn how easy it is to define and execute unit tests for your Cloud App.

7.1.1 Creating a unit test

Unit tests are part of your Fabasoft app.ducx project and must be defined in a Fabasoft app.ducx object model file.

We're going to define the unit tests for your Cloud App in a file named `unittests.ducx-om`. So go ahead and create a new Fabasoft app.ducx object model file with the name `unittests.ducx-om` and add a software component reference to the `FSCDUCXUNIT@1.1001` software component. Then add an import declaration for software component `FSCDUCXUNIT@1.1001`.

For each unit test, you have to define an instance of `FSCDUCXUNIT@1.1001:UnitTest` as illustrated by the example presented later on.

For the sake of brevity, we will only include a single unit test in this book. However, you can obtain the full source code for our Cloud App (including all the unit tests) from the public Subversion repository (see chapter "Retrieving code samples from the public Subversion repository" on page 151).

As a rule of thumb, you should define one unit test for each and every use case in your project that is not implemented as a virtual application. Virtual applications and all the other stuff that needs a GUI must be covered by Fabasoft app.test tests.

Generally, it's a good idea to have one unit test per use case, but you may also combine logically related use cases into a single unit test. This is exactly what we'll be doing in our example, where we will cover the `RecordTrip` use case and the `InitTrip` use case with the same unit test.

A unit test is comprised of three Fabasoft app.ducx expressions:

- The `localsetup` expression allows you to define a setup expression for creating and initializing the test containers for your unit test.
- In the `test` expression, cover all the possible test cases for the use case to be tested by your unit test. This is the "main body" of the unit test.
- The `localteardown` expression allows you to define a cleanup expression for deleting the test containers used by your unit test.

Keep the following important things in mind when writing a unit test:

- Add assertions to your expression code using the `%%ASSERT` directive to check if your test conditions are fulfilled. If at runtime a condition is not met, the assertion yields `false` and the unit test fails.
- Make sure that every branch of your use case implementation is covered, also the error handlers. You can use try-catch blocks to raise errors and test if they are handled correctly.
- Don't forget to cover the corner cases and also try to anticipate obscure scenarios.
- Clean up after yourself and delete all of the test objects created by your unit test.

The following example shows how to define a unit test.

Example

unittests.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCDUCXUNIT@1.1001;
  instance UnitTest UnitTestRecordTrip {
    localsetup = expression {
      //-------------------------------------------------------------------------------
      // Create and initialize test data
      //-------------------------------------------------------------------------------
      Logbook logbook = #Logbook.ObjectCreate();
      logbook.objname = "My Logbook";
      logbook.logvehicleid = "AG76282";
      SelectionContext selctx = {
        selview = #logtriplogs,
        selcontainer = :>logbook
      };
      #TV.TV_SELECTIONCONTEXT = selctx;
      #TV.TV_SELECTIONCONTAINER = logbook;
      TripLog triplog = #TripLog.ObjectCreate();
      triplog.ShareObject(null, null, #logtriplogs, logbook);
      cootx.Commit();
      %%ASSERT(logbook != null);
      %%ASSERT(triplog != null);
      %%ASSERT(triplog in logbook.logtriplogs);
      %%ASSERT(triplog.objaclref == logbook);
      %%ASSERT(triplog.objaclobj == null);
    }
    test = expression {
      //-------------------------------------------------------------------------------
      // Test the "RecordTrip" and "InitTrip" use cases
      //-------------------------------------------------------------------------------
      Trip trip = null;
      try {
```

```
      triplog.RecordTrip(trip);
      %%FAIL;
    }
    catch (#ErrIncompleteTripInfo) {
    }
    triplog.InitTrip();
    %%ASSERT(triplog.trlnewtrip != null);
    %%ASSERT(triplog.trlnewtrip.trpstartmileage == null);
    %%ASSERT(triplog.trlnewtrip.trpdriver == coouser);
    %%ASSERT(triplog.trlnewtrip.trptype == #TermBusiness);
    triplog.trlnewtrip.trpdepartureat = coonow - 9*60*60;
    triplog.trlnewtrip.trparrivalat = coonow - 6*60*60;
    triplog.trlnewtrip.trpdepartureplace = "New Haven, CT";
    triplog.trlnewtrip.trpdestinationplace = "Boston, MA";
    triplog.trlnewtrip.trpstartmileage = 10000;
    triplog.trlnewtrip.trpendmileage = 10150;
    triplog.trlnewtrip.trppurpose = "Customer Visit";
    triplog.RecordTrip(triplog.trlnewtrip);
    cootx.Commit();
    %%ASSERT(count(triplog.trltrips) == 1);
    %%ASSERT(triplog.trltrips[0].trpmileage == 150);
    %%ASSERT(triplog.trltrips[0].trpduration == 3*60*60);
    %%ASSERT(triplog.trltrips[0].trpdriver == null);
    %%ASSERT(triplog.trltrips[0].trpdrivername == coouser.objname);
    %%ASSERT(triplog.trlnewtrip == null);
    triplog.InitTrip();
    %%ASSERT(triplog.trlnewtrip != null);
    %%ASSERT(triplog.trlnewtrip.trpstartmileage == 10150);
    %%ASSERT(triplog.trlnewtrip.trpdriver == coouser);
    %%ASSERT(triplog.trlnewtrip.trptype == #TermBusiness);
    triplog.trlnewtrip.trpdepartureat = coonow - 7*60*60;
    triplog.trlnewtrip.trparrivalat = coonow - 5*60*60;
    triplog.trlnewtrip.trpdepartureplace = "Boston, MA";
    triplog.trlnewtrip.trpdestinationplace = "Salem, MA";
    triplog.trlnewtrip.trpendmileage = 10170;
    triplog.trlnewtrip.trppurpose = "Customer Visit";
    try {
      triplog.RecordTrip(triplog.trlnewtrip);
      %%FAIL;
    }
    catch (#ErrDepartureBeforeLastArrival) {
    }
    triplog.trlnewtrip.trpdepartureat = coonow - 4*60*60;
    triplog.trlnewtrip.trparrivalat = coonow - 5*60*60;
    try {
      triplog.RecordTrip(triplog.trlnewtrip);
      %%FAIL;
    }
    catch (#ErrArrivalBeforeDeparture) {
    }
    triplog.trlnewtrip.trpdepartureat = coonow - 4*60*60;
    triplog.trlnewtrip.trparrivalat = coonow - 3*60*60;
    triplog.trlnewtrip.trpstartmileage = 9999;
    triplog.trlnewtrip.trpendmileage = 10170;
    try {
      triplog.RecordTrip(triplog.trlnewtrip);
      %%FAIL;
    }
    catch (#ErrStartMileageLastEndMileage) {
    }
    triplog.trlnewtrip.trpstartmileage = 10175;
    triplog.trlnewtrip.trpendmileage = 10170;
    try {
      triplog.RecordTrip(triplog.trlnewtrip);
```

```
        %%FAIL;
      }
      catch (#ErrStartMileageEndMileage) {
      }
      triplog.trlnewtrip.trpstartmileage = 10150;
      triplog.trlnewtrip.trpendmileage = 10170;
      triplog.RecordTrip(triplog.trlnewtrip);
      cootx.Commit();
      %%ASSERT(count(triplog.trltrips) == 2);
      %%ASSERT(triplog.trlnewtrip == null);
    }
    localteardown = expression {
      //-------------------------------------------------------------------------
      // Clean up test data
      //-------------------------------------------------------------------------
      logbook.ObjectDelete();
      cootx.Commit();
    }
  }
}
```

7.1.2 Running a unit test

Save and upload your changes into your Cloud App VDE (see chapter “Uploading your Cloud App into the Cloud Sandbox” on page 79) and point your browser to the Cloud Sandbox. Then log in using the “developer” account.

From within an administration tool, issue a search for the instance of object class “Unit Test” with the reference `UnitTestRecordTrip` and share it into the administration tool by selecting it in the list of search results and clicking “Next”.

Select “Run Test” from the context menu of the unit test to execute it. The dialog box depicted in Figure 54 shows the result of the execution.

Refer to [Faba11a] for further information on how to define and run unit tests.

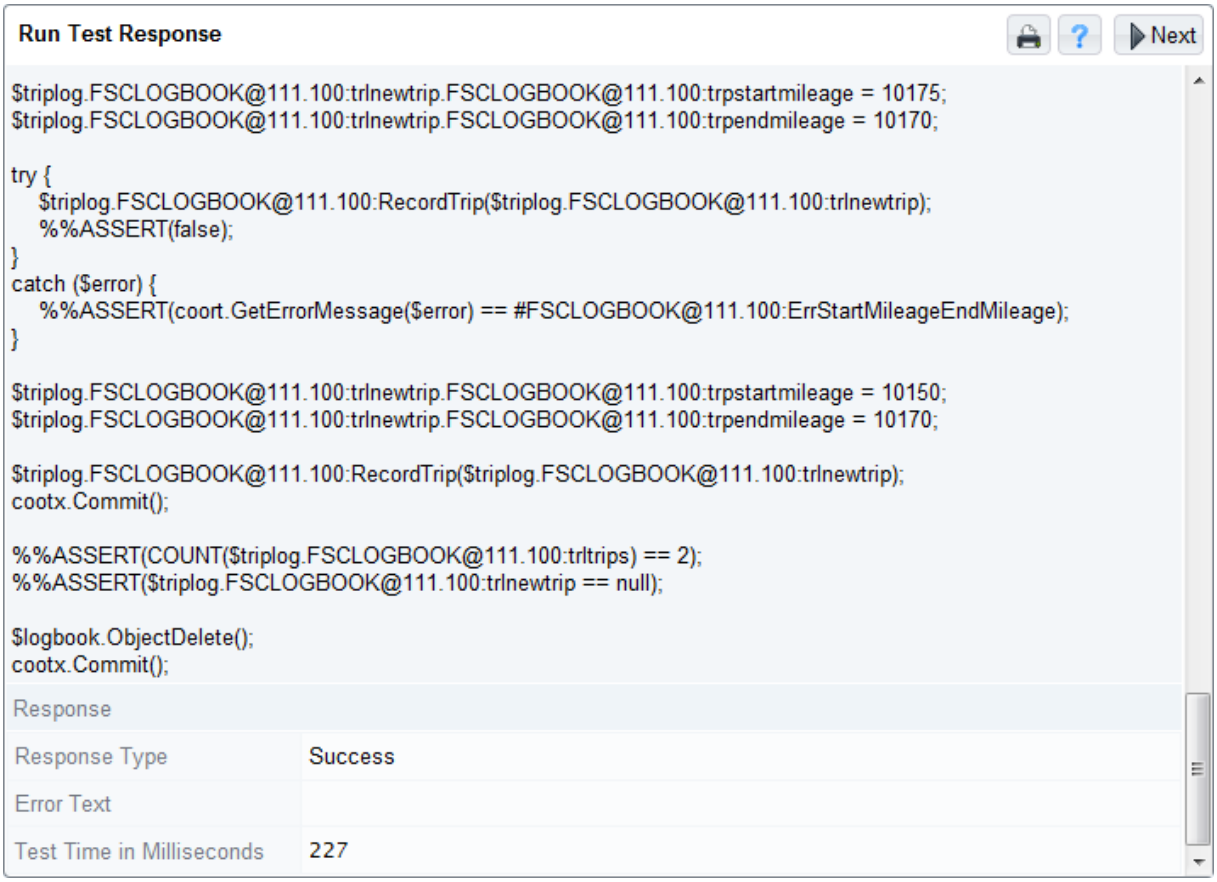


Figure 54: Executing a unit test

7.1.3 Creating and running a unit test group

If you have multiple unit tests for your Cloud App, you can combine them into a unit test group. This allows you to select “Run Test” from the context menu of the unit test group to execute the whole battery of unit tests that are part of the group.

The following example shows how to define a unit test group.

Example

unittests.ducx-om

```
objmodel FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import COODESK@1.1;
  import FSCDUCXUNIT@1.1001;
  instance UnitTest UnitTestInitTrip {
    …
  }
  instance UnitTest UnitTestRecordTrip {
    …
  }
  instance UnitTestGroup UnitTestGroupLogbook {
    tests = {
      UnitTestInitTrip,
      UnitTestRecordTrip
    }
  }
}
```

7.2 Creating and running Fabasoft app.test tests

Fabasoft app.test extends the testing possibilities of unit tests and enables you to conduct GUI testing. To test your Cloud App, Fabasoft app.test operates just like real human users would do when interacting with it.

While your unit tests ensure that your use cases can be executed with different input parameters and various settings, they don't allow you to interact with the GUI components of your Cloud App. This is where Fabasoft app.test takes over as it complements your unit tests with automated test sequences imitating a human user working with your Cloud App.

In addition, Fabasoft app.test provides a range of customizable reports tailored to your needs.

7.2.1 Installing Fabasoft app.test Studio primo

Before you can start creating your first test you need to download and install Fabasoft app.test Studio primo, which is available free of charge at `http://www.apptest.com/apptest/downloads`.

Refer to the Fabasoft app.test online help (available at `http://help.apptest.com`) and [Faba11b] for in depth instructions on how to install and configure Fabasoft app.test.

7.2.2 Importing the Fabasoft app.test project

In the next step, you have to import the skeleton project that was automatically created for your Cloud App from the Subversion repository.

In contrast to Fabasoft app.ducx, Fabasoft app.test does not allow you to import a project straight from a Subversion repository. Therefore, you have to check out the Fabasoft app.test project for your Cloud App into a folder using a Subversion client.

If you are using TortoiseSVN, select "SVN Checkout" from the context menu of Microsoft Windows Explorer to open the dialog box depicted in Figure 55.

In the *URL of repository* field enter the URL of your Cloud App in the Subversion repository and add the `test` folder to point it to the Fabasoft app.test project root. For our example, the full URL to the Fabasoft app.test project is:

```
https://folio.fabasoft.com/svn/apps/FCB798DAF91D3911AB30860F534FADBA/trunk/FSCLOGBOOK/test
```

Note: Refer to the chapter "Creating the development project" on page 42 for further information on how to retrieve the Subversion URL of your Cloud App.

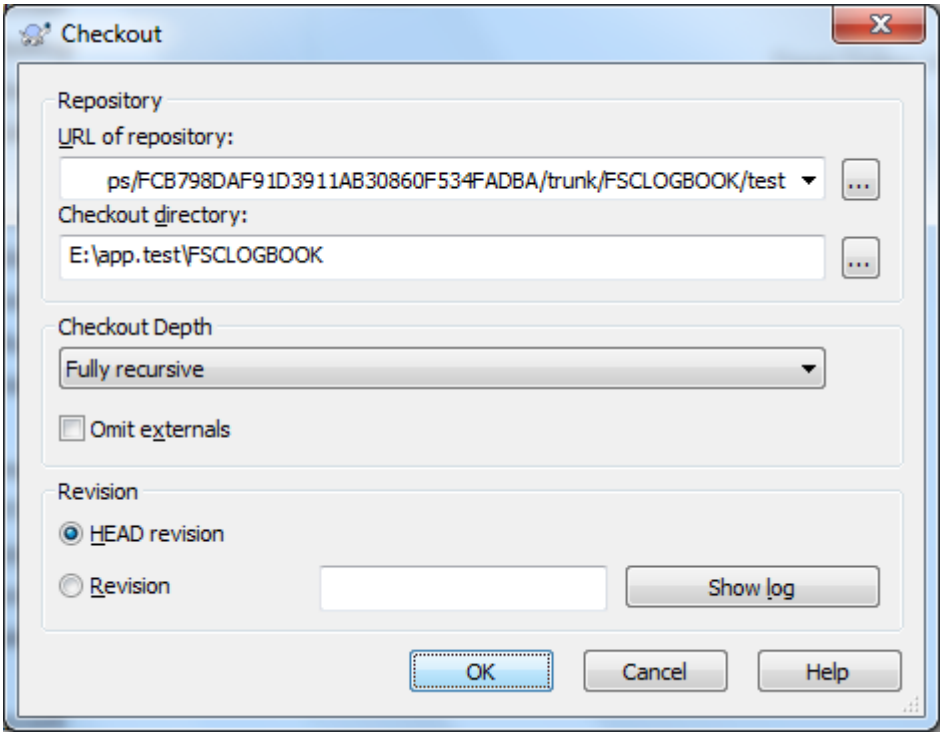


Figure 55: Checking out the Fabasoft app.test project

After checking out the project, start Fabasoft app.test and point the workspace folder to the folder where you checked out your project. Then select “File” > “Import” to bring up the dialog box depicted in Figure 56. In the “Import” dialog box, select “Existing Fabasoft DUCXtest or Fabasoft app.test Projects into Workspace” from the “Fabasoft app.test” branch in the *Select an import source* field and click “Next”.

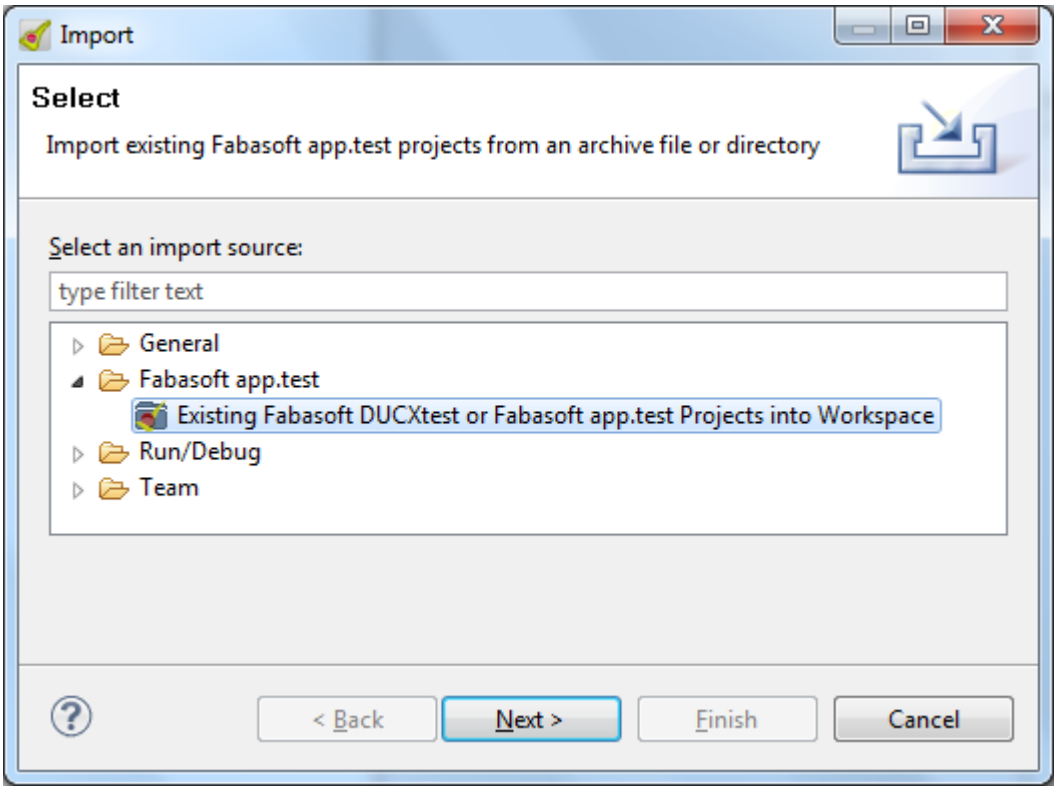


Figure 56: Importing an existing Fabasoft app.test project

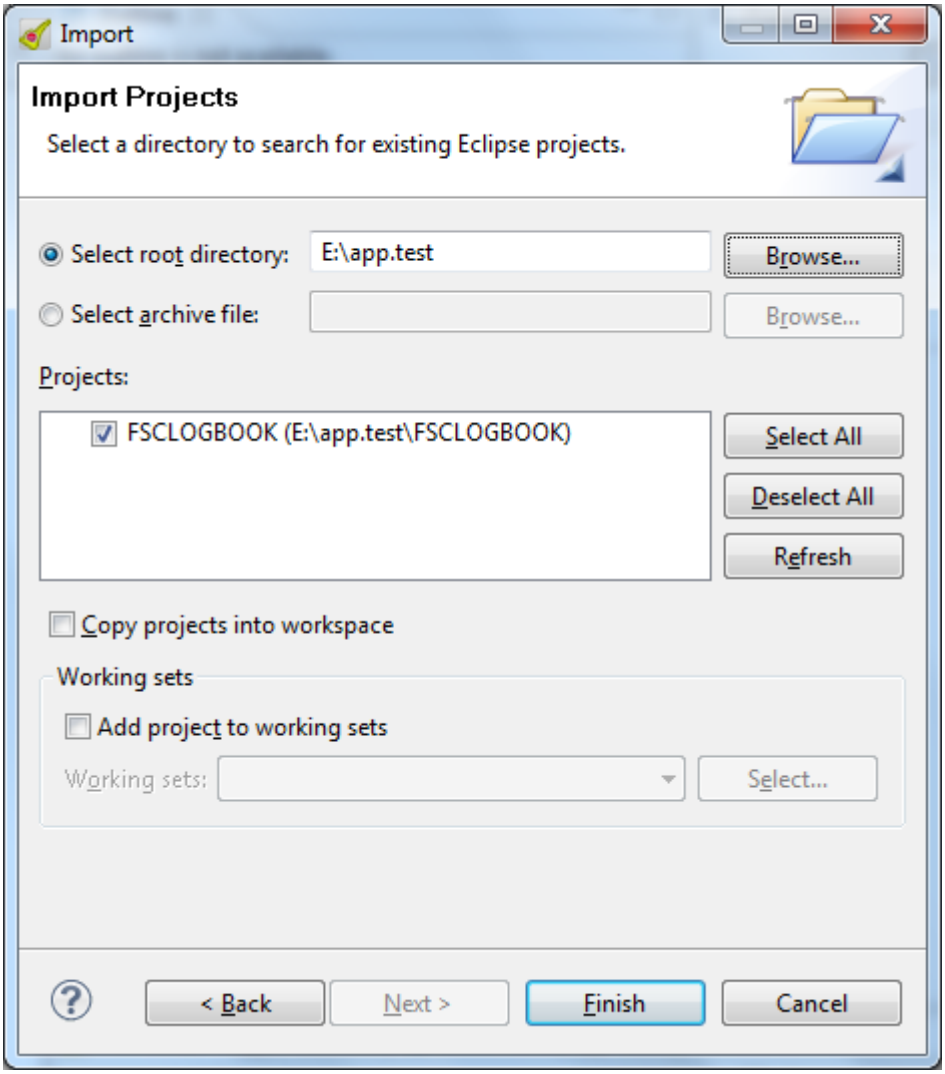


Figure 57: Selecting the Fabasoft app.test project to import

In the “Import Projects” dialog box shown in Figure 57, select the folder to where you checked out your Fabasoft app.test project in the *Select root directory* field and click “Finish” to finish the import.

7.2.3 Importing the common test sequences and use cases

After importing the skeleton project for your Cloud App into Fabasoft app.test, follow the instructions in [Faba11c] to import the common test sequences and use cases into your project.

This library of predefined test sequences and use cases tailored to Fabasoft Folio Cloud contains reusable test sequences that you can call in your tests so you don’t need to create sequences for common use cases (such as the login of a test user) by yourself.

7.2.4 Creating a test

Let’s create our first Fabasoft app.test test!

To get started, it would be a good idea to watch the video tutorials available at `http://www.apptest.com/community/get-started`. These video tutorials are definitely the easiest and most convenient way to get you up to speed with Fabasoft app.test.

In a nutshell, the general structure of the Fabasoft app.test project for your Cloud App is as follows:

- The project should contain a single test file invoking all of the sequences belonging to it.
- Break down your test into as many logical sequences as you wish. Sequences should not contain any executions (e.g. “clicks”) though.
- Each sequence of your test may contain multiple use cases that are responsible for carrying out the actual “work” (e.g. creating a logbook and filling out its properties).

For the sake of brevity, we will only focus on the absolute basics of creating a test. For further information refer to the Fabasoft app.test online help (available at `http://help.apptest.com`) and [Faba11b].

First of all, we need to create a test file. To create a test file, select “File” > “New” > “Test” and enter `logbook.ducx-test` in the *File name* field of the dialog box that is opened and click “Finish” (see Figure 58).

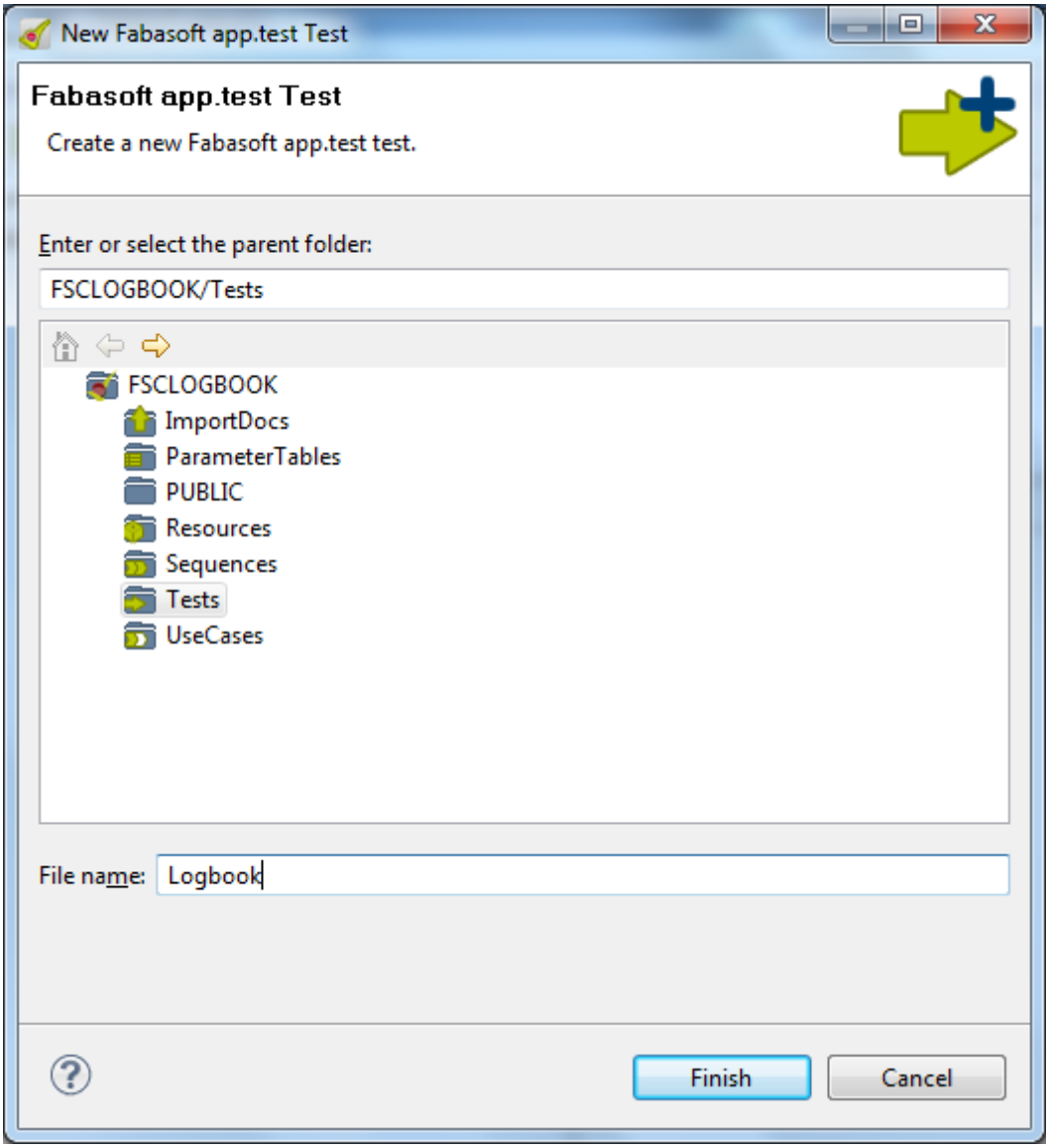


Figure 58: Creating a new test

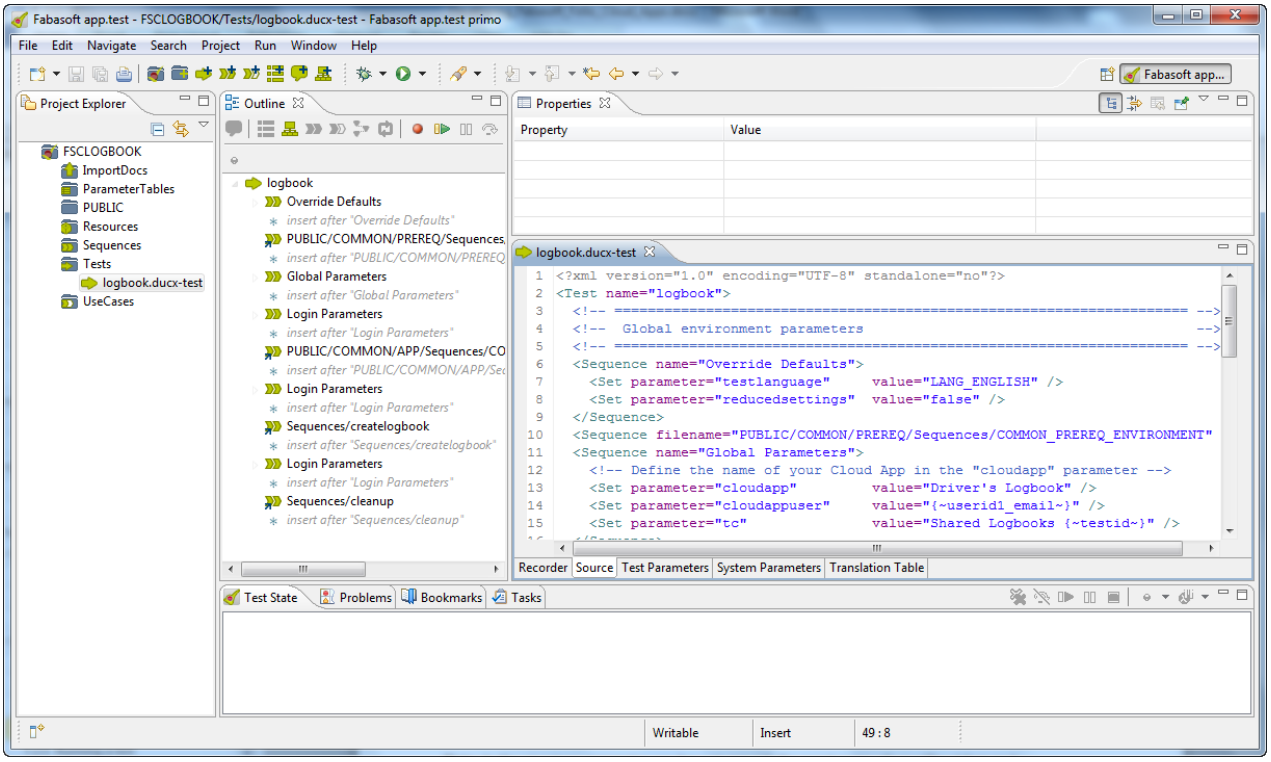


Figure 59: Editing the source code of a test

After creating the file, open the source code of the test by selecting the “Source” view of the `logbook.ducx-test` file, copy the code from the following example into the source code and save it.

The code is responsible for doing some initialization work before calling the `createlogbook.ducx-seq` test sequence, which we’re going to create in a subsequent step. Part of the initialization work is to acquire a license for your Cloud App for the test user.

Note: In the `cloudapp` parameter you have to provide the exact name of your Cloud App as it appears in the GUI (i.e. the name you defined in the `mlnames.lang` file).

Example

logbook.ducx-test

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Test name="logbook">
  <!-- ================================================================== -->
  <!--  Global environment parameters                                     -->
  <!-- ================================================================== -->
  <Sequence name="Override Defaults">
    <Set parameter="testlanguage"     value="LANG_ENGLISH" />
    <Set parameter="reducedsettings"  value="false" />
  </Sequence>
  <Sequence filename="PUBLIC/COMMON/PREREQ/Sequences/COMMON_PREREQ_ENVIRONMENT" />
  <Sequence name="Global Parameters">
    <!-- Define the name of your Cloud App in the "cloudapp" parameter -->
    <Set parameter="cloudapp"         value="Driver's Logbook" />
    <Set parameter="cloudappuser"     value="{~userid1_email~}" />
    <Set parameter="tc"               value="Shared Logbooks {~testid~}" />
  </Sequence>
  <!-- ================================================================== -->
  <!--  License Cloud App for Wanda Carney (as administrator)             -->
  <!-- ================================================================== -->
  <Sequence name="Login Parameters">
    <Set parameter="testuser"         value="{~useradminid0~}" />
```

```
    <Set parameter="testuser_pwd"     value="{~useradminid0_pwd~}" />
    <Set parameter="testuser_email"   value="{~useradminid0_email~}" />
    <Set parameter="appname"          value="{~cloudapp~}" />
    <Set parameter="contactemail"     value="{~cloudappuser~}" />
  </Sequence>
  <Sequence filename="PUBLIC/COMMON/APP/Sequences/COMMON_APP_LicenseAppForUser"
            username="{~testuser~}" password="{~testuser_pwd~}" />
  <!-- ================================================================== -->
  <!--  Create a new logbook as Wanda Carney                              -->
  <!-- ================================================================== -->
  <Sequence name="Login Parameters">
    <Set parameter="testuser"         value="{~userid1~}" />
    <Set parameter="testuser_pwd"     value="{~userid1_pwd~}" />
    <Set parameter="testuser_email"   value="{~userid1_email~}" />
  </Sequence>
  <Sequence filename="Sequences/createlogbook"
            username="{~testuser~}" password="{~testuser_pwd~}"/>
  <!-- ================================================================== -->
  <!--  Clean up as Wanda Carney                                          -->
  <!-- ================================================================== -->
  <Sequence name="Login Parameters">
    <Set parameter="testuser"         value="{~userid1~}" />
    <Set parameter="testuser_pwd"     value="{~userid1_pwd~}" />
    <Set parameter="testuser_email"   value="{~userid1_email~}" />
  </Sequence>
  <Sequence filename="Sequences/cleanup"
            username="{~testuser~}" password="{~testuser_pwd~}" />
</Test>
```

Now create a sequence file by selecting “File” > “New” > “Sequence” and enter `createlogbook.ducx-seq` in the *File name* field of the dialog box and click “Finish”. After that, copy the source code from the following example into the `createlogbook.ducx-seq` file.

Example

createlogbook.ducx-seq

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Sequence name="logbook">
  <!--  Login -->
  <Set parameter="portal"             value="{~deskportal~}" />
  <UseCase filename="PUBLIC/COMMON/INITIALIZE/UseCases/COMMON_INITIALIZE_Login" />
  <!-- Activate Cloud App -->
  <Set parameter="appname"            value="{~cloudapp~}" />
  <Set parameter="appactivate"        value="true" />
  <UseCase filename="PUBLIC/COMMON/APP/UseCases/COMMON_APP_ActivateAppForUser" />
  <!-- Create team room -->
  <Set parameter="portal"             value="{~deskportal~}" />
  <Set parameter="tc"                 value="{~tc~}" />
  <UseCase filename="PUBLIC/COMMON/TESTCONTAINER/UseCases/COMMON_TESTCONTAINER_
                     CreateTC_Teamroom" />
  <!-- Invoke the use case for creating a logbook in the team room -->
  <Set parameter="portal"             value="{~deskportal~}" />
  <Set parameter="tc"                 value="{~tc~}" />
  <UseCase filename="UseCases/createlogbook" />
</Sequence>
```

In the next step, create a use case file by selecting “File” > “New” > “Use Case” and enter `createlogbook.ducx-case` in the *File name* field of the dialog box and click “Finish”. Then click the “Start/Stop Recorder” button in the “Outline” view to bring up the dialog box depicted in Figure 60.

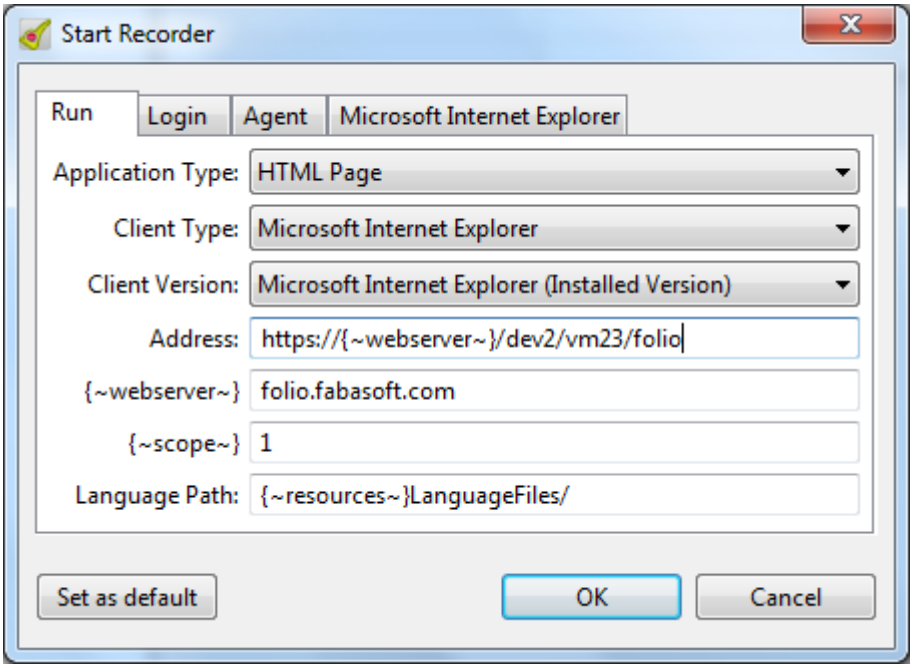


Figure 60: Starting the Fabasoft app.test recorder

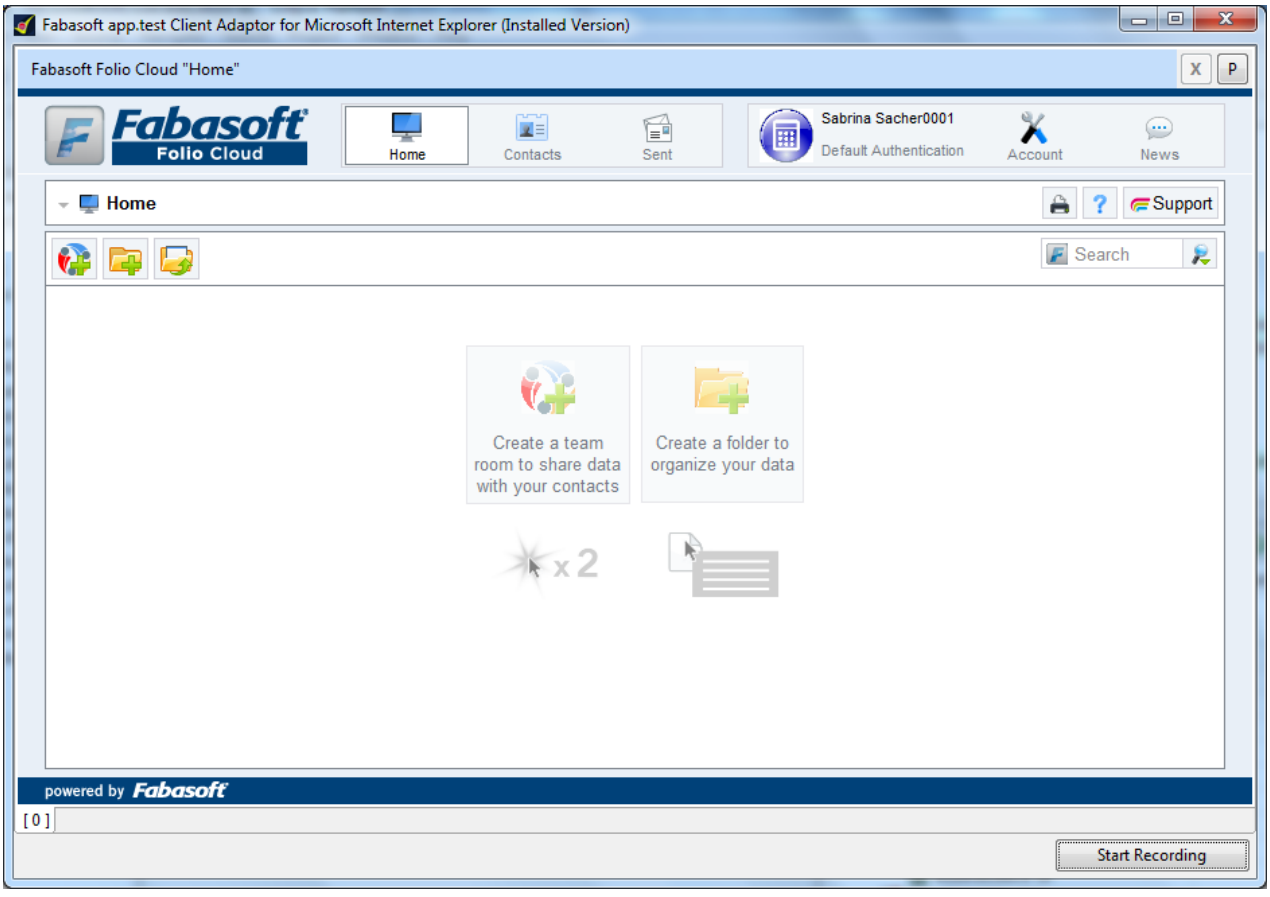


Figure 61: Working with the Fabasoft app.test recorder

In the *Address* field on the “Run” page, enter the URL of your Cloud Sandbox. You can also use the `{~webserver~}` variable when defining the URL, as illustrated by the example.

On the *Login* page, enter the user name and password of the user you want to use for creating your test. For our example, we enter `carney0001` in the *{~username~}* field and the password we defined for the test users in the *{~password~}* field (see chapter “Working with the Cloud App VDE” on page 35). Then click “OK” to start the recorder.

In the Fabasoft app.test recorder depicted in Figure 61, click the “Start Recording”’ button to start recording your interactions with the Cloud Sandbox. Once in recording mode, every mouse click and every push of a button is recorded and transformed into the corresponding statement, which is appended to the `createlogbook.ducx-case` file until you stop recording by clicking “Stop Recording”.

In order to keep the example concise, we’ll just carry out a few steps:

- Create a new team room and enter the team room by double-clicking on it. In our `createlogbook.ducx-seq` sequence file, we’re using a common use case for creating a new team room. Therefore, we don’t need to record the steps for creating the team room.
- Click the “Start Recording” button to start recording the executed steps.
- Create a new logbook named “Team Logbook” inside of the team room.
- Enter “TEAM1” in the *Vehicle ID* property of the logbook.
- Click “Next” to save and close the logbook.

When you’re done, click the “Stop Recording” button and switch to the source view of the `createlogbook.ducx-case` file, which should look like the following example.

Example

createlogbook.ducx-case

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<UseCase name="createlogbook">
  <Execution action="Click" location="PAGES.Main.Contents.BUTTONBAR.New"/>
  <Execution action="Click" location="PAGES.Main.What do you want to create?.All"/>
  <Execution action="Click" location="PAGES.Main.What do you want to create?[0].
                                      Logbook"/>
  <Execution action="Click" location="PAGES.Main.BUTTONBAR.Next"/>
  <Execution action="Set" location="PAGES.Main.Name" value="Team Logbook"/>
  <Execution action="Set" location="PAGES.Main.Vehicle ID" value="TEAM1"/>
  <Execution action="Click" location="PAGES.Main.BUTTONBAR.Next"/>
</UseCase>
```

In the next step, we’ll have to do some editing to make our use case compatible with the requirements of the Continuous Integration (CI) environment (see chapter “Continuous integration environment” on page 156) and to give it the finishing touches.

The main aspects of creating robust test use cases are discussed in great detail in [Faba11c], so we’ll limit ourselves to the most crucial things to keep in mind:

- Use variables for any values you’re planning to use repeatedly. Most importantly, use variables for the names of your test containers (e.g. the team room and the logbook) and include the `testid` variable in the names.
- At the end of the use case, return to the starting point. In our example this means going back to the “Home” screen.

The following example shows the source code of the `createlogbook.ducx-case` file after incorporating all of the mentioned points.

Example

createlogbook.ducx-case

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<UseCase name="createlogbook">
  <!-- ========================================================================= -->
  <!--  Parameters                                                               -->
  <!-- ========================================================================= -->
  <Set parameter="tmpportal"        value="{~portal~}" />
  <Set parameter="tmptc"            value="{~tc~}" />
  <Set parameter="tmplogbook        value="Team Logbook {~testid~}" />
  <!-- Set defaults-->
  <Set parameter="tmpportal"        value="{~deskportal~}"
    if='"{~tmpportal~}"=="NULL" || "{~tmpportal~}"==""' />
  <!-- ========================================================================= -->
  <!--  Create logbook within team room                                          -->
  <!-- ========================================================================= -->
  <Execution action="Click" location="PORTALS.{~tmpportal~}" />
  <Execution action="Click" location="BREADCRUMBS[1]"/>
  <!-- Enter team room -->
  <Set parameter="tmptcexists" location='PAGES.Main.CONTROLS.Contents["{~tmptc~}"]'
       eval="Exists"/>
  <Validation ok='"{~tmptcexists~}"=="true"' />
  <Execution action="Doubleclick" location='PAGES.Main.CONTROLS.
                                            Contents["{~tmptc~}"]' />
  <!-- Create logbook -->
  <Execution action="Click" location="PAGES.Main.Contents.BUTTONBAR.New"/>
  <Execution action="Click" location="PAGES.Main.What do you want to create?.All"/>
  <Execution action="Click" location="PAGES.Main.What do you want to create?[0].
                                      Logbook"/>
  <Execution action="Click" location="PAGES.Main.BUTTONBAR.Next"/>
  <Execution action="Set" location="PAGES.Main.Name" value="{~tmplogbook~}"/>
  <Execution action="Set" location="PAGES.Main.Vehicle ID" value="TEAM1"/>
  <Execution action="Click" location="PAGES.Main.BUTTONBAR.Next"/>
  <!-- Return to "Home" screen -->
  <Execution action="Click" location="BREADCRUMBS[1]"/>
</UseCase>
```

Now that we've got you started, it's up to you to improve your test and enrich it with additional use cases to cover all the functionality of your Cloud App not already covered by unit tests. Of course, it doesn't hurt if your Fabasoft app.test tests also cover parts of your code that are already covered by unit tests.

Don't forget to clean up when you're done (i.e. create a sequence file named `cleanup.ducx-seq` and delete all the objects created by your use cases in a cleanup use case) to leave the test environment in a clean state.

The examples presented in this chapter should have given you a rough idea of how to create test use cases. For detailed information and a full language reference refer to the Fabasoft app.test online help (`http://help.apptest.com`), [Faba11b] and [Faba11c].

Also, the complete set of tests for our sample is available to you in the public Subversion repository of Fabasoft (`https://folio.fabasoft.com/svn/public`). For further information refer to the chapter "Retrieving code samples from the public Subversion repository" on page 151.

7.2.5 Running a test

To run your test, select the `logbook.ducx-test` file in Project Explorer of Fabasoft app.test, open the context menu and select “Run” to bring up the dialog box shown in Figure 62. Then click “OK” to start the test.

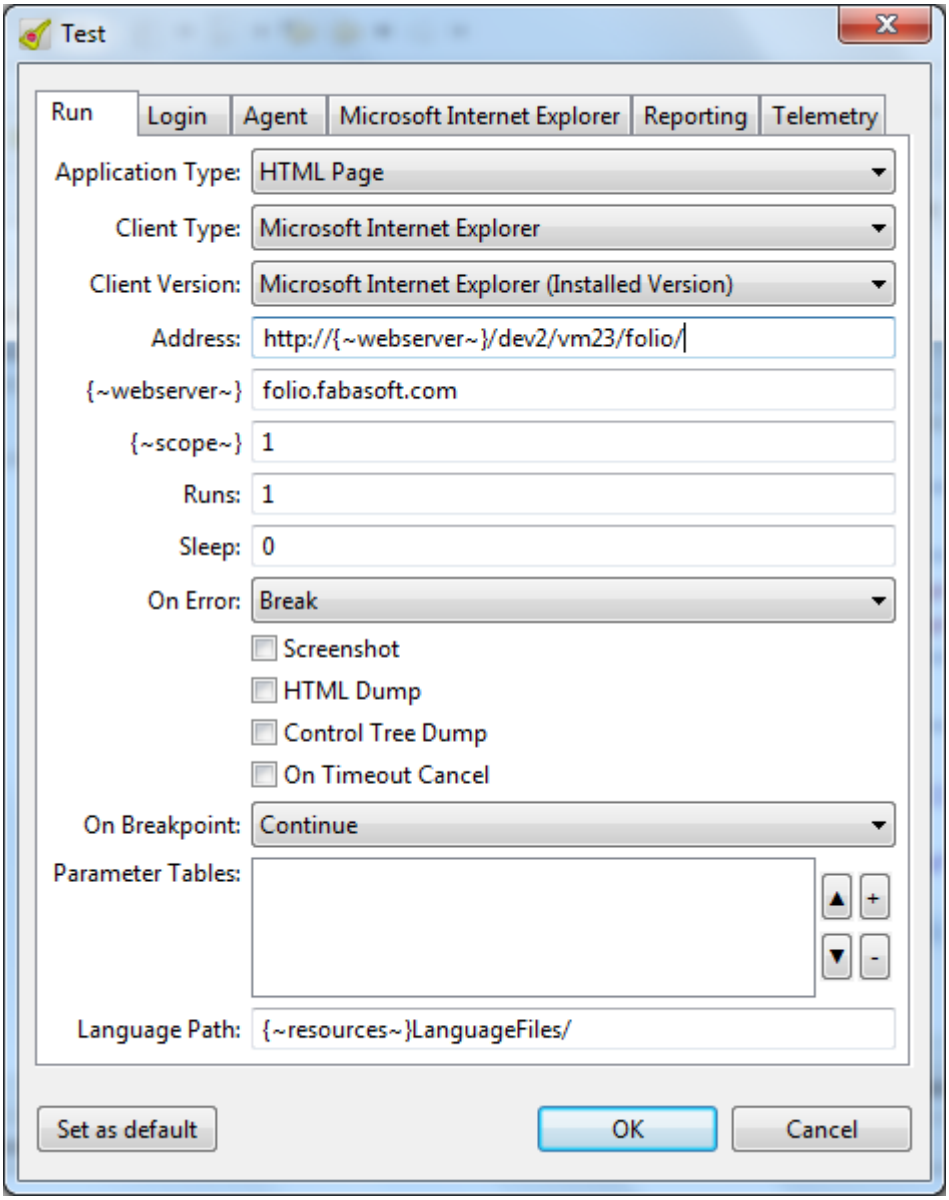


Figure 62: Running a test

Don’t close the window that is opened as this will abort the execution of your test. At the end of the test, a summary will be displayed in the “Test State” view.

For further information on running a test refer to the Fabasoft app.test online help (`http://help.apptest.com`), [Faba11b] and [Faba11c].

7.3 Checking and improving the coverage of your tests

As you already know, one of your main objectives is to reach 100 % code coverage. But how can you find out which parts of your code are already covered by your tests?

Well, Fabasoft app.ducx includes a built-in coverage plug-in that makes it a breeze to determine the coverage ratio for your Cloud App.

Did you already notice the “Coverage” view displayed by default in the right bottom panel of Eclipse? If it’s not open for some reason, select “Window” > “Show View” > “Other”, select “Coverage” from the “Fabasoft app.ducx Coverage” branch and click “OK”.

These are the steps to record coverage information:

- To start a coverage session, click the “Start Coverage Session” button (to the left of the “Hide/Show Coverage” button).
- The web browser is launched and you are prompted to log in to your Cloud Sandbox.
- During a coverage session, all of your actions and clicks (and also the actions carried out by unit tests and Fabasoft app.test tests) are recorded by the coverage plug-in.
- When you stop the coverage session by clicking the “Stop Coverage Session” button, the recorded information is downloaded to Eclipse and you can drill down in your code to find the parts that have not been covered by your tests yet.

Coverage only applies to Fabasoft app.ducx expression code. Lines covered by a test are displayed in green whereas lines that have not been covered are displayed in red. Also, the number of times a line was visited during a coverage session is displayed.

Note: During a coverage session, every interaction with your Cloud Sandbox is recorded. When you run a Fabasoft app.test test against your Cloud Sandbox while a coverage session is active, the actions carried out by the test are also recorded.

Remember that coverage information is not user-specific. Every action carried out by any user in your Cloud Sandbox is taken into account for the coverage ratio.

To achieve a complete picture of the combined coverage ratio for your Cloud App, we suggest the following approach:

- Start a new coverage session by clicking the “Start Coverage Session” button.
- Log in using the “developer” user account.
- From within an administration tool, execute all of your unit tests.
- Switch to Fabasoft app.test Studio primo and run your test.
- Wait until your Fabasoft app.test test finishes.
- Click the “Stop Coverage Session” button.
- Examine the coverage information displayed in the “Coverage” view (see Figure 63).

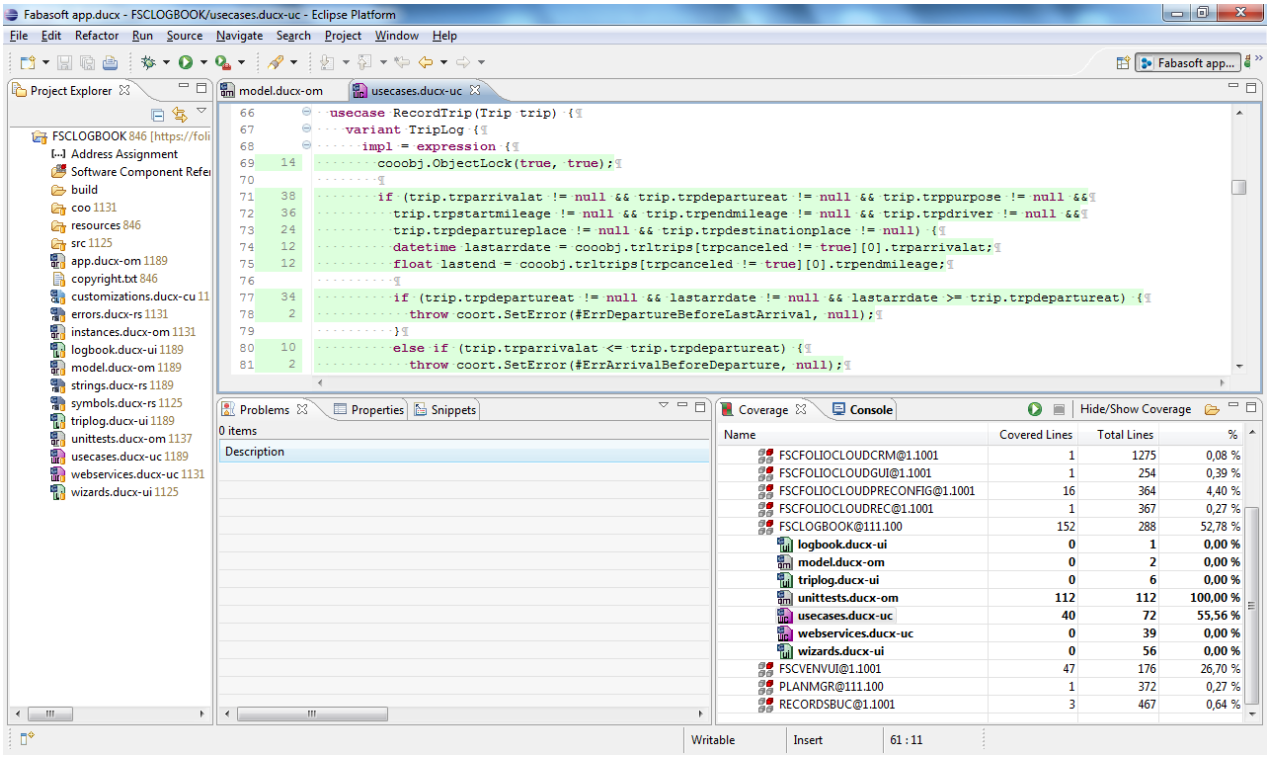


Figure 63: Checking the coverage

Getting help, code samples and support

8 Getting help, code samples and support

Got stuck somewhere along the road?

No need to worry!

For Cloud App developers, getting help is easy!

8.1 Help and documentation

The following resources should be your first resort when you need help:

Fabasoft Folio Cloud:

- Online help
- Quick tours: `http://www.foliocloud.com/quick-tour`
- Frequently asked questions: `http://www.foliocloud.com/faq`
- Community web site: `http://www.foliocloud.com/community`

Fabasoft app.ducx:

- Online help
- White papers [Faba11a], [Faba11e], [Faba11i]

Fabasoft app.test:

- Online help: `http://help.apptest.com`
- White papers [Faba11b], [Faba11c]

Fabasoft app.telemetry:

- Online help: `http://www.apptelemetry.com/documentation`

In addition, the “Fabasoft Folio Cloud” team room contains additional information about the release cycle, new features of Fabasoft Folio Cloud as well as links to additional documentation and other useful materials. Refer to the chapter “Fabasoft Folio Cloud update cycle” on page 26 for more information on the “Fabasoft Folio Cloud” team room.

Finally, the Fabasoft Folio Cloud developer web site at `http://developer.foliocloud.com` is another great resource for Cloud App developers, including a dedicated developer forum.

8.2 Retrieving code samples from the public Subversion repository

More often than not, a nifty code snippet tells more than a thousand words. That’s why we host a public Subversion repository full to the brim with code samples and even the complete source code of full-blown Cloud Apps.

Using a Subversion client you can access our public Subversion repository with the repository URL `https://folio.fabasoft.com/svn/public`. You can log in with your Fabasoft Folio Cloud credentials.

To import a Cloud App sample project into your Eclipse workspace, select “Import” from the “File” menu of Eclipse. In the “Import” dialog box, expand the “SVN” branch, select “Project from SVN” and click “Next” (see Figure 20 on page 46).

In the “Checkout from SVN” dialog box depicted in Figure 21 on page 47, enter the URL `https://folio.fabasoft.com/svn/public` and your Fabasoft Folio Cloud credentials in the *User* and *Password* fields of the *Authentication* box, then click “Browse” to log in to the Subversion repository.

In the “Select Resource” dialog box depicted in Figure 22 on page 48, navigate to the Cloud App project you want to import and select the “dev” branch underneath it. Then click “OK” to return to the “Checkout from SVN” dialog box and click “Finish” to proceed with the import of your project from Subversion.

8.3 Getting support from Fabasoft and the community

The Fabasoft Folio Cloud developer forum (see `http://developer.foliocloud.com`) allows you to get support from fellow Cloud App developers and Fabasoft experts.

For general Fabasoft Folio Cloud issues refer to the Fabasoft Folio Cloud support web site at `http://www.foliocloud.com/support`.

8.4 Staying up to date

Things are moving fast in Fabasoft Folio Cloud. Every month, we add tons of new features and continuously strive to improve existing functionality. Therefore, it’s crucial for Cloud App developers to stay up to date.

The “What’s New” documents published by Fabasoft in the “Fabasoft Folio Cloud” team room (see chapter “Fabasoft Folio Cloud update cycle” on page 26) after every Fabasoft Folio Cloud update should be your first resource for getting all the latest information about new features, any changes that might have a potential effect on your Cloud App and other development-related stuff.

Moreover, the Fabasoft Cloud Developer Conference (CDC) is a great way to learn about new features and techniques, connect with fellow Cloud App developers and get advice from the Fabasoft Cloud App development experts.

For further information on the Fabasoft CDC go to `http://www.foliocloud.com/cdc`.

Releasing your Cloud App

9 Releasing your Cloud App

Before your Cloud App is allowed to go live, it must pass a sophisticated release process to ensure it is fit for use in Fabasoft Folio Cloud.

In this chapter, we describe the release process and how you can submit your Cloud App to Fabasoft for review.

9.1 About the release process

Figure 64: The Cloud App release process

Upon submitting your Cloud App for review, the release process depicted in Figure 64 is started. It comprises the following steps:

- Plain CI environment tests: The Fabasoft app.test tests and unit tests for your Cloud App are executed in the Plain CI environment, which is an environment similar to your Cloud Sandbox.
- Full CI environment tests: If your Cloud App passes the Plain CI environment tests, it is deployed into the Full CI environment, which is identical to the production environment. Again, your Fabasoft app.test tests and unit tests are executed, and your Cloud App is checked for potential side effects on other Cloud Apps.
- Code review: After passing the Full CI environment tests, a manual review of the source code of your Cloud App is conducted by a Fabasoft reviewer.

If your Cloud App passes the review, it's good to go live with the next scheduled update of Fabasoft Folio Cloud.

For further information about the release process and the deadlines you have to meet for your Cloud App to be included in the next scheduled Fabasoft Folio Cloud update refer to [Faba11d].

9.2 Submitting your app for review

Make sure to have all your ducks lined up correctly before submitting your Cloud App to Fabasoft, because the release process might take a couple days to complete and you don't want to waste time and miss the next update window just because of a few missing comments. Therefore, review the list of deliverables in the chapter "What you need to do to get your Cloud App deployed" on page 29 before continuing.

When you're confident that your Cloud App meets all the requirements defined by Fabasoft, log in to Fabasoft Folio Cloud, select the release object you've created within the development project for your Cloud App and select "Submit" from the context menu to submit your Cloud App for review (see Figure 65).

Note: Remember to pick a price tier for your Cloud App before submitting it. For further information refer to the chapter "Reaping the profits" on page 159.

A branch of your source code is automatically created in the Subversion repository for the submitted release and the release process is kicked off.

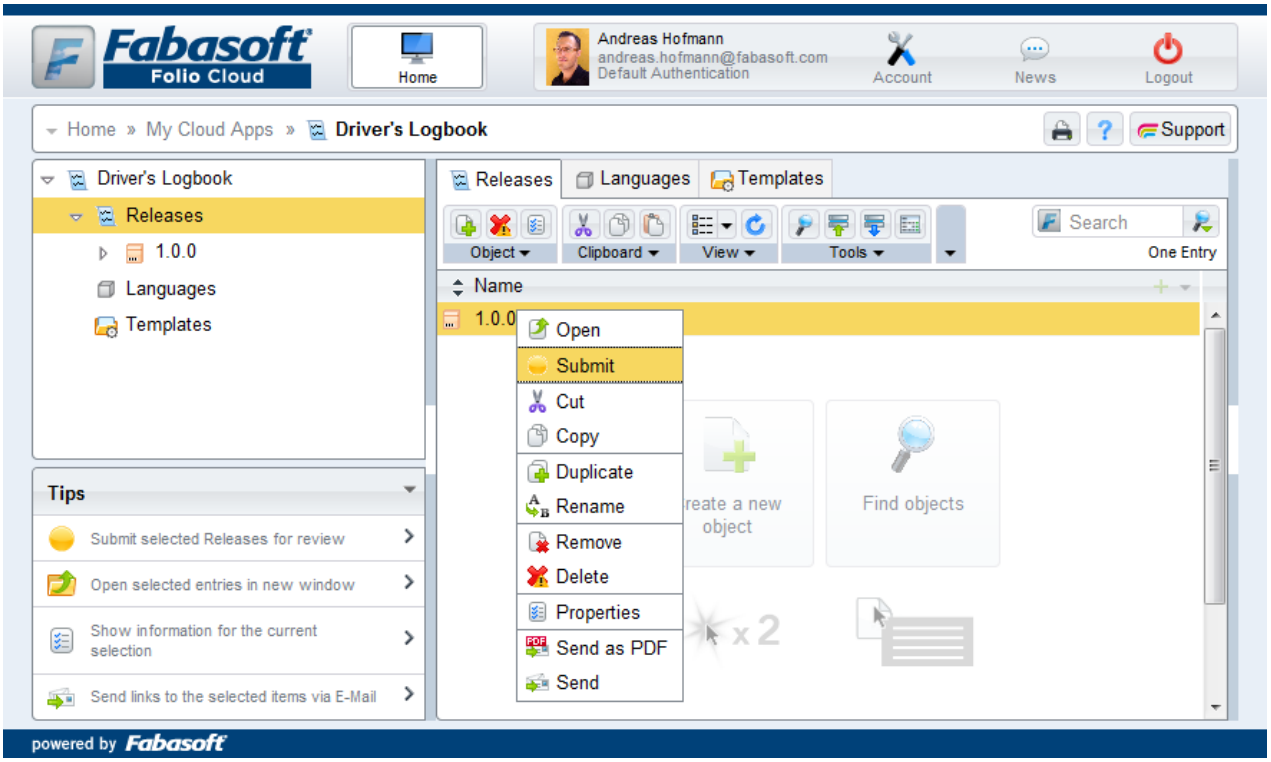


Figure 65: Submitting your Cloud App for review

9.3 Continuous integration environment tests

The continuous integration environment tests in the Plain CI environment and the Full CI environment are automatically scheduled and executed.

If your Cloud App passes the integration tests, it will be forwarded to a Fabasoft reviewer for manual review of the source code.

In case of errors, the review process is aborted and an error report is generated and published in the release object for the submitted release of your development project.

9.4 Code review

At Fabasoft, take code quality extremely seriously. Therefore, every submitted Cloud App must go through a thorough manual code review by a specially trained Fabasoft expert.

If any issues are detected during the code review, an issue report is created and made available to you in the release object for the submitted release.

9.5 Getting feedback

You will be notified via e-mail about the outcome of the release process. Moreover, a detailed report of the review results is published in the release object for the submitted release (see Figure 66).

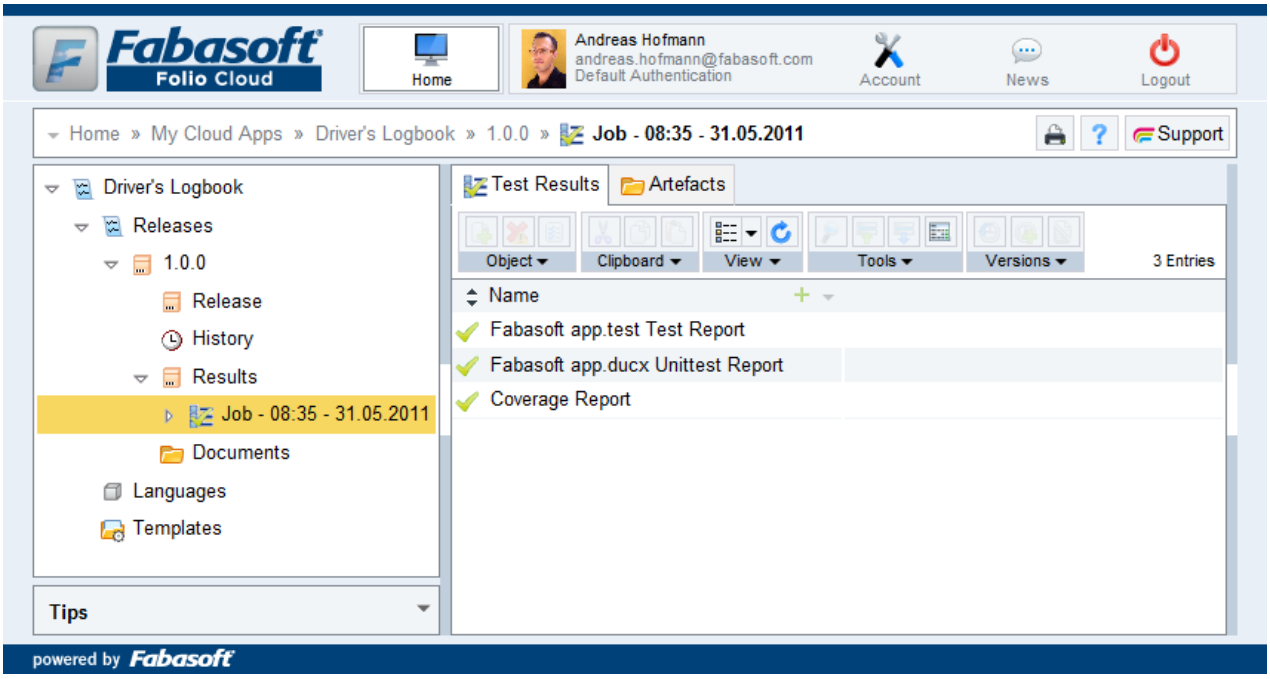


Figure 66: Results of the release process

If your Cloud App passes the review, it will be included in the next scheduled update of Fabasoft Folio Cloud.

Note: If you submitted your Cloud App after the submission deadline for the next schedule update, it will be included in the update after the next scheduled update. For further details refer to [Faba11d].

However, if your Cloud App doesn't successfully complete the release process, it will not be deployed into Fabasoft Folio Cloud. In this case, carefully review the report published in the release object, straighten out the detected issues and resubmit your Cloud App.

Reaping the profits

10 Reaping the profits

Let's talk business!

Fabasoft Folio Cloud is based on a revenue sharing model that is simple and transparent: As Cloud App developer, you get 50 % of the net proceeds of every sold subscription to your Cloud App. Basically, every sale of a Cloud App subscription generates recurring revenue for you, because users don't buy a perpetual license but a three month or one year subscription to your Cloud App.

In addition, you can charge users for using premium features within your Cloud App or provide services such as premium support at a fee using so-called Activity Points. Again, you get 50 % of the net proceeds of every Activity Point spent within the context of your Cloud App.

Be sure to read the fine print in the developer agreement for all the details about the revenue sharing model and the Activity Points of Fabasoft Folio Cloud
(see `http://www.foliocloud.com/developeragreement`).

10.1 About price tiers

Fabasoft does not allow you to attach an arbitrary price tag to your Cloud App. Instead, we require you to pick a predefined price tier from a list of available price tiers provided by Fabasoft.

A price tier defines the price of a Cloud App subscription for a given period of time. More precisely, it defines the price for a three month subscription and a one year subscription.

For further information about price tiers refer to the Fabasoft Folio Cloud developer agreement
(see `http://www.foliocloud.com/developeragreement`).

10.2 Defining the stuff related to billing

All the information regarding pricing and billing must be defined on the "Statement" page of your development project in Fabasoft Folio Cloud (depicted in Figure 67 on page 160):

- If your Cloud App will not be free of charge, enter your bank account details in the properties *Account Holder*, *IBAN* and *BIC*.
- If you have a tax number, set the *Liable to VAT* property to "Yes" and enter your tax number in the *VAT ID* property.
- If you want to receive your monthly app sales statements by e-mail, set the *Send App Statements by E-Mail* property to "Yes". Otherwise, you will not be notified when a new app sales statement is available.
- Create a new entry in the *Price for App* property and enter the reference of your Cloud App in the *App Reference* property (i.e. `AppFSCLOGBOOK`). In the *Required Folio Cloud Edition*, select the edition of Fabasoft Folio Cloud you require for your Cloud App and in the *Price Tier* property select one of the preconfigured price tiers.

10.3 Activity Points

Activity Points allow you to charge users for using premium features or services (e.g. premium support) within your Cloud App.

Users can purchase Activity Points in the Fabasoft Folio Cloud App Store and then use their points for consuming premium features or services within context of Cloud Apps.

For each premium feature or service you want to provide at a charge, you have to define a custom Activity Point type and assign a price tier to it.

The example in this chapter outlines how to define a custom Activity Point type for your Cloud App, which you can then use to debit points from a user's point balance for using a premium feature of your Cloud App.

For our example, assume that we provide a premium feature in your Cloud App that allows users to have trip logs validated by an external auditing service.

First, we define a custom Activity Point type named `AppPointTripLogValidation` in a Fabasoft app.ducx object model file. Then you have to assign a price tier to your Activity Point type by adding an entry in the *Price for Activity Points* property of your development project in Fabasoft Folio Cloud (see Figure 67).

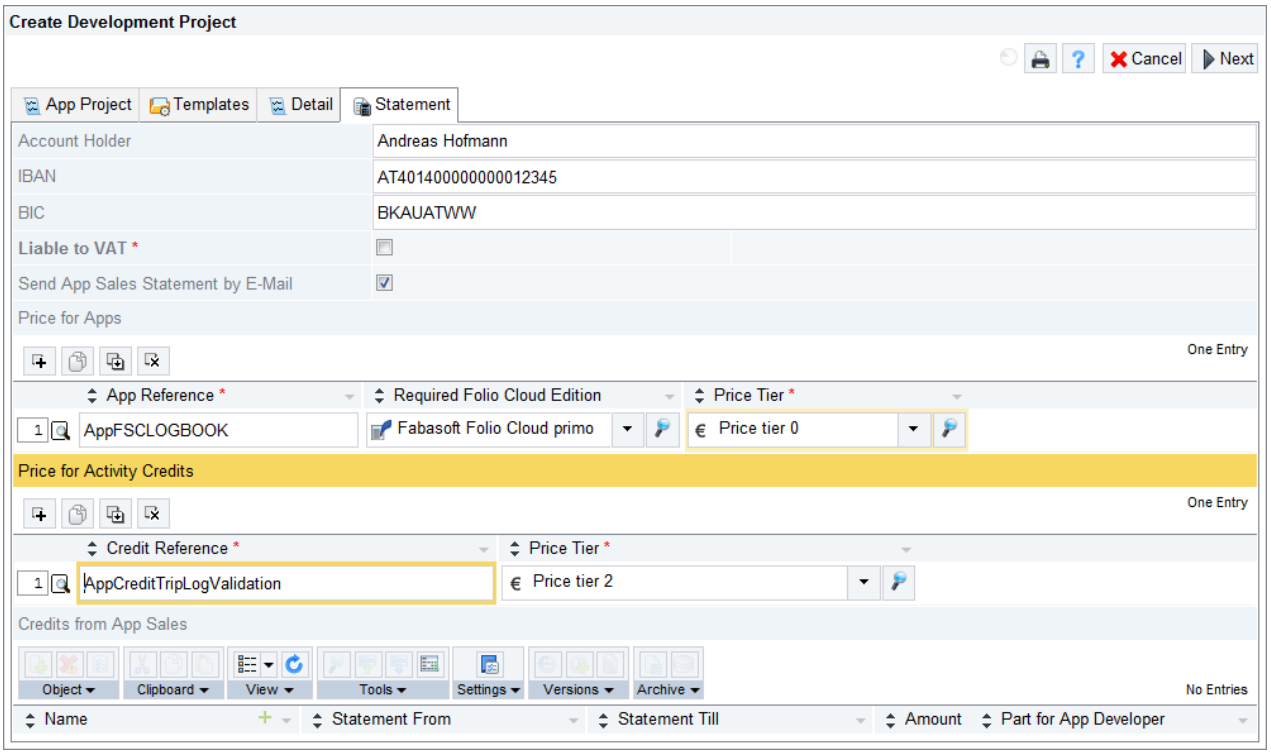


Figure 67: The "Statement" page of the development project

In the virtual application that implements the premium feature, we invoke the virtual application `FSCFOLIOCLOUDPAYMENT@1.1001:ObjectDebitActivityPoint` to check the user's point balance and then deduct a point from their balance.

Example

instances.ducx-om

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
  import FSCTERM@1.1001;
  import COOATTREDIT@1.1;
  …
  instance ActivityPoint AppPointTripLogValidation {
    symbol = SymbolAppFSCLOGBOOK;
    acapps = AppFSCLOGBOOK;
  }
}
```

validation.ducx-uc

```
usecases FSCLOGBOOK@111.100
{
  import COOSYSTEM@1.1;
```

```
import FSCVAPP@1.1001;
import FSCVENV@1.1001;
import FSCVIEW@1.1001;
import FSCFOLIOCLOUDPAYMENT@1.1001;
menu usecase ValidateTripLogWizard on selected or container {
  variant TripLog {
    impl = application {
      expression {
        integer availablepoints;
        // Check if the user has enough points to proceed. If parameter 3 is set to
        // "false", only a balance check is performed but no points are deducted.
        ->ObjectDebitActivityPoint(#AppPointTripLogValidation, null, false,
          &availablepoints);
        // Proceed only if the user has enough points
        if (availablepoints > 0) {
          boolean dovalidation = false;
          // TODO: Do something (e.g. ask the user if she wants to send this trip log
          // for validation by an external auditing service)…
          if (dovalidation) {
            // If the user decided to proceed with the validation, deduct a point
            // of type "AppPointValidationRequest" from her point balance
            ->ObjectDebitActivityPoint(#AppPointTripLogValidation, cooobj.objname,
              true, null);
            // TODO: Continue with validation
          }
        }
      }
      …
    }
  }
}
```

Note: Whenever a Cloud App attempts to deduct a point from a user's point balance, the user is asked to confirm the transaction (see Figure 68).

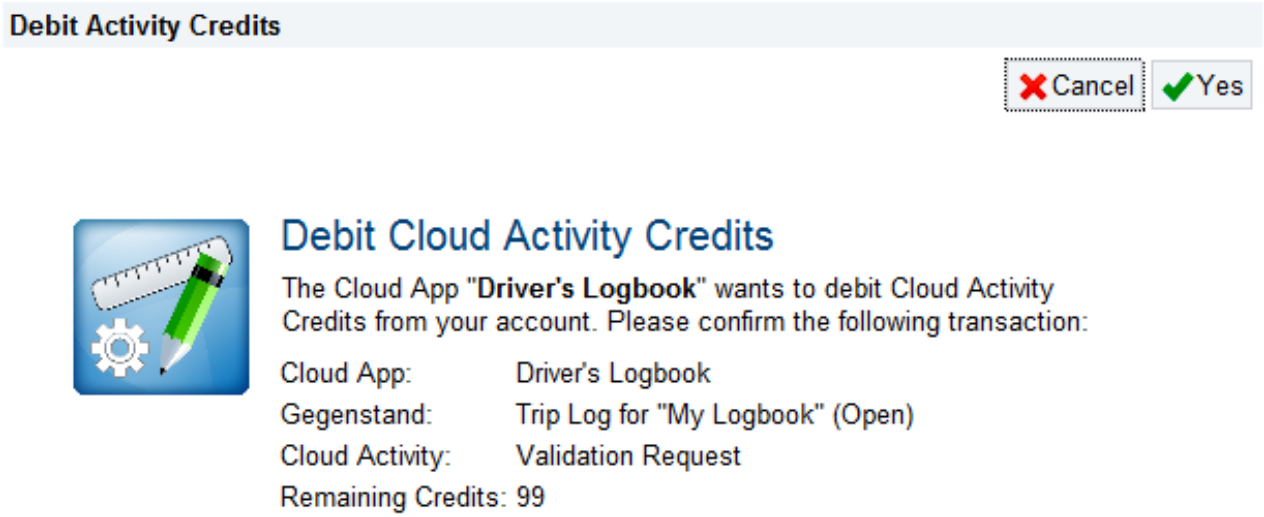


Figure 68: Request to debit Activity Points

10.4 Getting your money

Just sit back and relax, because your share of the proceeds from Cloud App subscription sales and Activity Point purchases is automatically deposited into your bank account.

In addition, a detailed sales statement is generated and saved in the *App Sales Statements* property of your developer project.

For further details refer to the Fabasoft Folio Cloud developer agreement at
`http://www.foliocloud.com/developeragreement`.

Maintaining & improving your Cloud App 11

11 Maintaining and improving your Cloud App

In this chapter, we describe what you need to do in terms of maintenance and how you can release an update of your Cloud App.

11.1 Updating Fabasoft app.ducx projects using Eclipse

As the list of Fabasoft Folio Cloud features keeps growing and growing over time, you will also have to keep your Fabasoft app.ducx project up to date.

With every release of a Fabasoft Folio Cloud update, a plethora of new object classes and properties, use cases and applications, and dozens of other new component objects become available to you to use and incorporate into your Cloud App.

And even though we make every effort to minimize the impact on you and your Cloud App, you sometimes may be required to make some changes to your existing code in order to keep your Cloud App intact.

For example, if the reference of an object class provided by Fabasoft is changed for reasons of consistency, and your Cloud App uses this object class as a prerequisite, you will need to adapt your code before this change is released to the public with the next Fabasoft Folio Cloud update.

Note: Periodically check the “What’s New” documents (see chapter “Fabasoft Folio Cloud update cycle” on page 26) to find out about changes that might impact your Cloud App, so you can make necessary code modifications in time.

To keep your Cloud App up to date, carry out the following steps in Eclipse:

- Make sure that the Fabasoft app.ducx plug-in is up to date (see chapter “Updating the Fabasoft app.ducx plug-in” on page 34).
- In the “Project Explorer”, navigate to your Cloud App project.
- Open the context menu of the “Software Component References” folder and click “Update All References”.
- Open the “Project” menu and click “Clean”.

This way you get an overview of all warnings and errors in your project that may arise because of changes in Fabasoft Folio Cloud caused by the update. For more information about renamed, deleted and obsolete component objects consult [Faba11i].

11.2 Reacting to user feedback and providing customer support

The main platform for communicating with the subscribers of your Cloud App is the Fabasoft Folio Cloud forum at `http://www.foliocloud.com/support/forum`.

Once your Cloud App has been released, a special subsection will be created for your Cloud App in the forum and you are required to monitor and process the forum posts created in there.

If you want to provide premium support services to the users of your Cloud App, you can host and handle your own support system.

For further details refer to the Fabasoft Folio Cloud developer agreement at `http://www.foliocloud.com/developeragreement`.

11.3 Releasing an updated version

To create and release an updated version of your Cloud App, you have to follow these steps:

- Create a new release object: In your development project, create a new release object for the next version of your Cloud App and select “Start Release” from its context menu.
- Update your code, tests and documentation: Make all the desired changes to your source code, update your unit tests, Fabasoft app.test tests and the context-sensitive help.
- Release your update: Now just follow the same steps as you did previously to release the original version of your Cloud App. For further details refer to the chapter “Releasing your Cloud App” on page 155.

Easy peasy, isn’t it?

12 Glossary

12 Glossary

Access Control List (ACL)
Determines which user is granted which access rights for an object.

Activity Point
A pre-paid credit token that can be used for executing premium features or consuming services provided by a Cloud App.

Branch
A button in a dialog of a virtual application.

Cloud App
A self-contained application hosted in Fabasoft Folio Cloud.

Cloud Sandbox
The Fabasoft Folio Cloud installation that is part of your Cloud App VDE and allows you to develop and test your Cloud Apps.

Constraint
A rule for calculating or validating values, or for preventing invalid data entry into a property.

Continuous Integration (CI) Environment
An environment for running automated tests for quality control.

Dialog
An element of a virtual application for presenting a user interface to provide a means of communication between a user and a virtual application.

Fully Qualified Reference
A unique identifier for referring to a component object. The fully qualified reference consists of the software component prefix, followed by a colon and the reference of a component object, e.g. `COOSYSTEM@1.1:objname`.

Keyword
A reserved sequence of characters.

Reference
An identifier for referring to a component object.

Software Component Prefix
The part of a fully qualified reference that refers to the software component, e.g. `COOSYSTEM@1.1`.

Trigger
An action that is invoked when a predefined event occurs.

Virtual Development Environment (VDE)
A preconfigured development environment including software tools and services for supporting Cloud App development.

List of figures

13

13 List of figures

Bibliography and useful links

14 Bibliography and useful links

[ApSF11] **Apache Software Foundation**: “Apache Subversion”. URL: http://subversion.apache.org [Retrieved on March 8, 2011]

[Ecli11] **Eclipse Foundation**: “Eclipse Downloads”. URL: http://www.eclipse.org/downloads [Retrieved on February 18, 2011]

[Faba11a] **Fabasoft**: “White Paper: An Introduction to Fabasoft app.ducx”.

[Faba11b] **Fabasoft**: “White Paper: Fabasoft app.test“.

[Faba11c] **Fabasoft**: “White Paper: Creating automated tests for Cloud Apps with Fabasoft app.test”.

[Faba11d] **Fabasoft**: “Cloud Developer Information”. URL: http://www.foliocloud.com/cdi [Retrieved on June 1, 2011]

[Faba11e] **Fabasoft**: “White Paper: Parameters of Fabasoft Folio Controls“.

[Faba11f] **Fabasoft**: “White Paper: An Introduction to Fabasoft Folio”.

[Faba11g] **Fabasoft**: “White Paper: Installation and Configuration of Fabasoft Integration for CalDAV”.

[Faba11h] **Fabasoft**: “White Paper: Installation and Configuration of Fabasoft Integration for CMIS”.

[Faba11i] **Fabasoft**: “White Paper: Renamed, Deleted and Obsolete Component Objects”.

[IETF07] **IETF**: “RFC 4791 – Calendaring Extensions to WebDAV (CalDAV)”. URL: http://tools.ietf.org/html/rfc4791 [Retrieved on May 17, 2011]

[OASI10] **OASIS**: “Content Management Interoperability Services (CMIS) Version 1.0”. URL: http://docs.oasis-open.org/cmis/CMIS/v1.0/os/cmis-spec-v1.0.html [Retrieved on May 17, 2011]

[Orac11a] **Oracle**: “Download Free Java Software”. URL: http://www.java.com/download [Retrieved on February 18, 2011]

[Orac11b] **Oracle**: “How to Write Doc Comments for the Javadoc Tool”. URL: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html [Retrieved on March 23, 2011]

[ScrA09] **Scrum Alliance**: “What Is Scrum?”. URL: http://www.scrumalliance.org/learn_about_scrum [Retrieved on February 22, 2011]

ISBN 978-3-902495-28-0

Fabasoft www.fabasoft.com

Learn how to build your own Cloud Apps for Fabasoft Folio Cloud, the secure online document management and collaboration platform by Fabasoft! This book is a comprehensive introduction to Cloud App development with Fabasoft app.ducx, Fabasoft app.test and Fabasoft app.telemetry. It gives clear step-by-step instructions on how to efficiently create slick Cloud Apps in just a couple hours. Written in non-technical language, the book presents a wide array of examples and resources for further reading, which makes it the ideal Cloud App development companion for both novice and expert developers.