

FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Angewandter Informatik

Documentation of the Application Development Process
using the Toro Web Framework by means of a Developer
Tutorial

Dokumentation des Anwendungsentwicklungsprozesses mit
dem Toro Webframework anhand eines Entwicklertutorials

Author: Sebastian Graf Henckel von Donnersmarck
Supervisor: Prof. Dr. Florian Matthes
Advisor: Dr. Thomas Büchner
Submission Date: 15. September 2008

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master's thesis only supported by declared resources.

Place, Date

Signature

Abstract

The use of the Toro – web application framework is explained through a tutorial in the context of developing a web-application. The tutorial-application is a simplified subsystem of a larger system, also developed in Toro. The development and architecture of the larger system is discussed, the detailed features of Toro and their application are shown through the smaller system.

Keywords: Toro, introspection, tutorial, web-application

Content

Abstract	3
Content	4
Preface	7
1 Overview	8
1.1 Architecture	9
2 SmsParty	11
2.1 Business Aspects	11
2.1.1 The SMS standard	11
2.1.2 Gateway Providers	12
2.1.3 SMS-Application Examples	13
2.1.4 Customers.....	15
2.2 Technical Aspects	17
2.2.1 Original Approach.....	17
2.2.2 Data Model.....	18
2.2.3 Messaging System	20
3 Tutorial	28
3.1 Goal	28
3.2 Installation	29
3.2.1 Installation – Basic	29
3.2.2 Installation - Expert.....	30
3.3 Instantiating a new project.....	35
3.4 Implementing Persistence – Assets and Properties.....	38
3.5 Implementing Persistence – DomainValueProperties	48
3.6 Implementing Persistence – The ‘Person’ Class.....	51
3.7 Implementing Persistence – Relationships between Assets	57
3.8 Implementing Persistence – Registering your Assets	60
3.9 Implementing Persistence – Persisting Data	61
3.10 Implementing Logic – Pageful Handlers	63
3.11 Implementing Logic – Pageless Handlers.....	67
3.12 Implementing Presentation – Print Substitutions.....	72
3.13 Implementing Logic – Setting Parameters	77
3.14 Implementing Presentation – Substitution-Functions and Template Substitutions.....	79

3.15	Implementing Presentation – Conditional Substitutions	80
3.16	Implementing Presentation – Editing-Forms for Assets	83
3.17	Implementing Logic – Queries on the Database	89
3.18	Implementing Presentation – ListSubstitutions	91
3.19	Implementing Logic – Configuration	95
3.20	Implementing Logic – extending Assets	103
3.21	Implementing Persistence – Deleting Assets.....	107
3.22	Implementing Logic – Checking User Authorization.....	109
3.23	Implementing Logic – Logging Out	111
	Appendix A: Listings	112
	References.....	148

Preface

The aim of the following master's thesis was twofold: I (as the writer) had a certain web-application I wanted to build, and Thomas Büchner, had just finished a new Persistence and webvisualization framework, Toro, that he was getting ready to release to the public, and wanted me to write a tutorial for new users. So the decision was made to have me build my web-application in Toro, and that I document the process in form of a tutorial. We soon realized that the scope of my web-application was too broad for a tutorial: Describing the construction of the whole application did not seem a good introduction: It would be too long, repetitive and include a lot of code not specific to Toro.

It was therefore decided to use a 'toy-application', a subset of the full web-application, as the Tutorial example. I would, however, never have been able to write a tutorial for Toro without going through the development of a whole application: Toro is an 'Introspective framework', and discussions with Thomas made me slowly understand the meaning of the introspective programming paradigm. As with object-orientation, the idea behind introspection is subtle, and its usefulness is not immediately apparent. When you begin, you are rather annoyed with a seemingly verbose and obtuse programming style, whose significance escapes you. It is only later on in the project that you see why things are done the way they are, and investments pay off.

What is introspection? Introspection is writing code not only to produce a program, but also to provide the framework (=Toro) and the Integrated Development Environment (=Eclipse) with information about this program. By empowering the framework and the IDE through this additional code (which seems useless at first), the programmer then has at his disposal tools and methods that could not work otherwise. Persistence is a good example: Mapping Java primitives and classes to a Database application is impossible without further information about the data. This additional information can be supplied through annotations (as is done in the Hibernate framework), or by overwriting methods in Anonymous Inner Classes. Toro chooses the latter approach.

At this point, it is not yet clear whether introspection really is the future. Java may not be an ideal language to implement introspection. Also, I am still unsure which of the two Java-mechanisms most adapted to pass on introspective information, AICs and Annotations, is better at the job. It took me quite some time to get used to the Anonymous Inner Class mechanism that Toro uses to supply the meta-information about persistent types, and had long discussions with Thomas trying to convince him annotations would be a better choice here. When Thomas had finally won me over, I could now no longer understand why the software configuration feature of Toro is implemented using Annotations. He is still arguing this point with me....

1 Overview

The thesis is larger in scope than the mere tutorial. The real-life application discussed in chapter (2) is cut down to size in the Tutorial, developed in chapter (3), so as not to overwhelm a newcomer and confuse him with irrelevant details. The tutorial aims at getting a programmer who knows nothing of the Toro framework to a point where he can use the framework as fast as possible. Excursions, spread throughout the tutorial, will discuss certain concepts that go beyond a mere 'how to'.

My aim was to develop an SMS (Short Messaging Service) dating game. The rules of the game are simple: Guests of a party (in a dance-club, for example) all wear large stickers with different numbers. If one guest, call her Alice, spots another guest whom she fancies (let's call him Bob), Alice sends an SMS with Bob's sticker number (e.g. 32) to a central phone number containing the following text: 'Love 32'.

At this point, nothing yet happens. If however, Bob were to fancy Alice as well, and send *her* number (e.g. 23) to the central phone number using the text 'Love 23', a 'Match' occurs. Both Alice and Bob are therefore informed of their mutual interest in one another through a SMS-Message to their respective mobile phones. Alice receives the message "You have a match! Guest 32 (Bob) also has an interest in you!". Bob, on the other hand, receives the message: "You have a match! Guest 23 (Alice) also has an interest in you!"

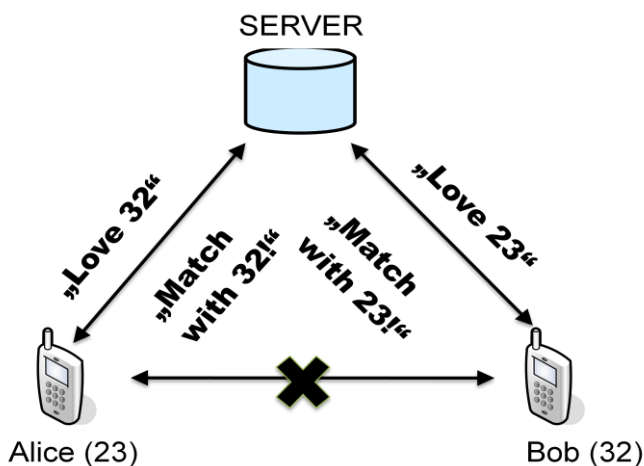


Figure 1: Simple Game Model

The point of the game obviously is to enable guests to anonymously chat each other up without fear of rejection. No direct communication happens between the two suitors, as indicated by the crossed out arrow in Figure 1). A detailed description of the different

Messages which can be sent to the Server, and the various responses these Messages trigger are left to Chapter (2.2.3)

1.1 Architecture

The application described above isn't really a web-application. Current SMS-Gateways (interfaces that allow SMS Messages to be sent and received by a computer) however work with URL (unique resource Locator) techniques that are well adapted to be accessed through a webvisualization framework such as Toro. But still, standing alone, this could not be considered to be a true 'web-application'. However, besides this SMS-application, an application has to be developed that permits a party administrator to generate, cancel, and keep track of the billing and general status of individual parties:

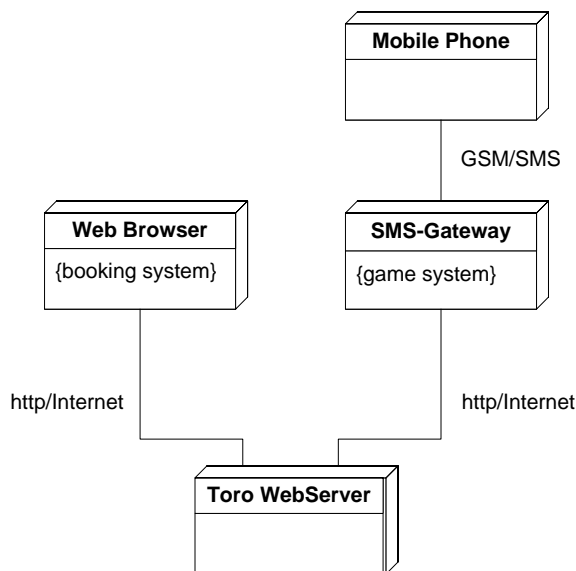


Figure 2: Deployment Diagram of SmsParty

As can be seen in Figure 2), the SmsParty application communicates with the party-organizer's browser to book and administrate parties. The actual game is played via mobile phones, which can also be accessed via http, but through the services of a SMS-Gateway, which translates URLs into SMS messages, and transmits incoming SMS by 'touching' call-back URLs.

So two applications were developed: One application, which we will call the "SMS-Engine" will receive, process and respond to the messages of party guests sent per mobile phone – this corresponds to the right hand side of Figure (2). The other applica-

tion, which I refer to by the name of “Booking-system” will permit an administrator to create new parties, open accounts for individual party-organizers, and keep track of billing. Both applications work on the same database (not shown).

2 SmsParty

As mentioned before, the thesis will be split into a part describing the real-life web-application, and a tutorial part, which will explain a reduced part of the real-life web-application in detail. This section deals with the real-life application.

2.1 Business Aspects

The web-application designed during the work on this thesis is a 'Premium SMS Service', which charges the customer for sending a message to the server. Although we will be dealing primarily with technical questions, the following chapters will discuss Business aspects of the designed application.

2.1.1 The SMS standard

Mobile phones capable of sending and receiving short messages have an unparalleled market penetration. The simplicity of the standard has helped it to become universally available. SMS is a part of the GSM standard for mobile communication, whose success can be estimated by looking at the number of SIM-Cards in the hands of the German public:

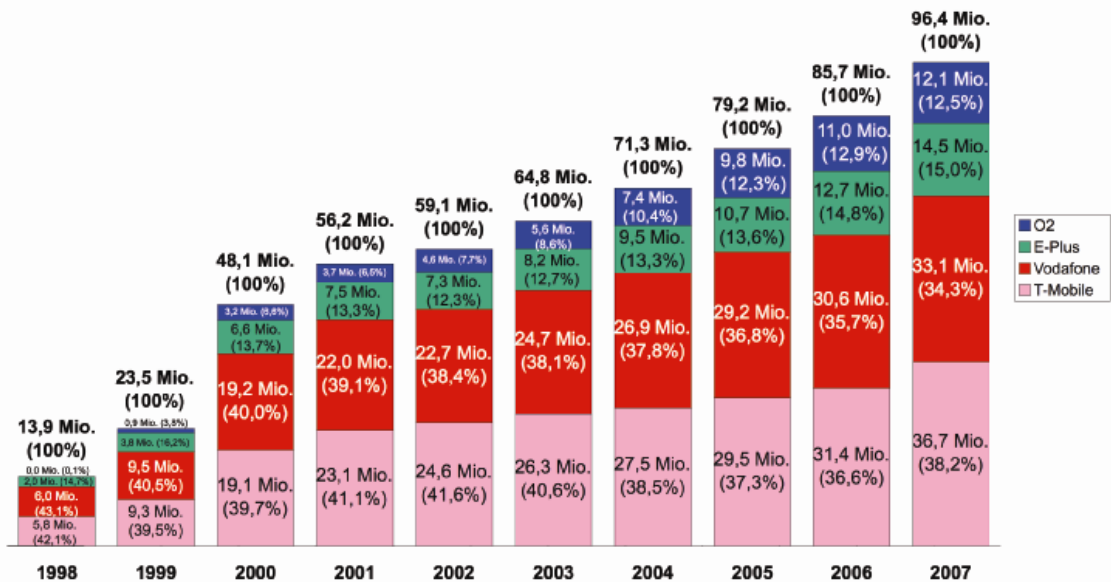


Figure 3: Number of SIM-Cards by Mobile Phone Company in Germany (Estimate) [DV07]

As can be seen, already by 2006, the number of SIM-Cards in Germany had overtaken the population in Germany (Population: 82 Million). More than half of the Non-Voice Mobile Communications Revenue comes from SMS Communications. The share of

MMS (Multimedia Messaging service) and Data is growing, but still not dominant. If at all possible, applications requiring only a simple text-interface should therefore still be implemented using the SMS standard: Market penetration is virtually 100%, no software has to be installed, and compatibility will not be an issue.

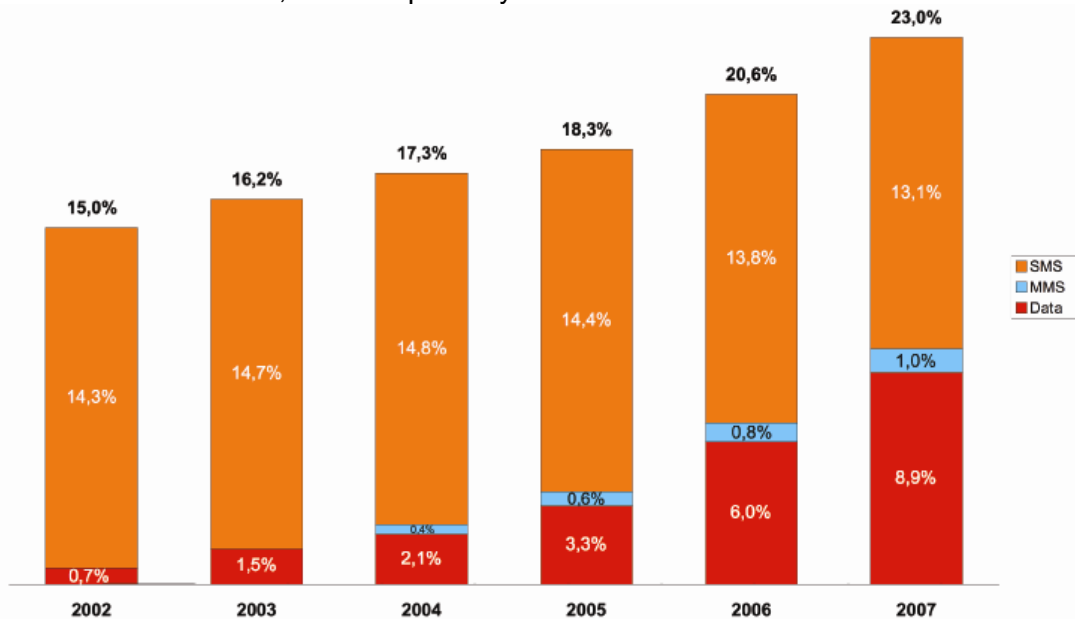


Figure 4: Non Voice Market Share of the Mobile Communications Market (Estimate) [DV07]

2.1.2 Gateway Providers

Mobile phone companies¹ have outsourced gateway-marketing to several small companies: Purchasing virtual mobile-numbers to receive SMS, mass postings of SMS etc. are not marketed directly: No trace of such services can be found on the websites of the mobile-phone companies, and several direct requests about such services failed to provide a response. However, there are very many sub-providers who do offer this service.

Incoming SMS schemes can be classified according to the following Categories:

- Premium:

Sending a SMS to the Gateway-Number entails a fee charged to the account of the sender in addition to the normal price of a SMS. This fee is split between the Gateway Provider and the Gateway Taker. The split can be as high as 75% for the Gateway Provider (mes.mo GmbH) or lower than 50% (Synapsy Mobile Networks GmbH).

- Non-Premium:

Sending an SMS to the Gateway-Number only costs the sender the normal amount for sending an SMS.

¹ in Germany

Every Scheme is either Premium or Non-Premium. In addition to this classification, the type of number to which the SMS are sent can belong to one of three different classes:

- Dedicated Keynumber:

The most expensive solution: a five digit short phone number is reserved exclusively for reception of SMS messages for the specified application. The advantage is a short, memorable number.

- Shared Keynumber with Keywords

Here, the application is also accessed via a five digit short phone number, but the number is potentially used by several applications. To distinguish the applications a keyword has to be prefixed to the SMS, in front of the actual message. The advantage, from the perspective of the application provider, is price: As the number can be shared, it is not as expensive to rent.

- Virtual GSM – Number

The cheapest solution: A 4+7 digit (virtual) GSM number is reserved for the reception of SMS-Messages. Keywords to share the number between different applications are not needed, but the number is long.

The following table gives an overview of typical fixed and variable costs for a non-premium SMS Service as charged by the German Provider OpenIT GmbH :

Scheme	Setup Charge	Monthly Charge	Reception Cost/SMS
Dedicated Key-Number	2000,00 €	850,00 €	0,005 €
Shared Key-Number	200,00 €	200,00 €	0,005 €
Virtual GSM-Number	200,00 €	80,00 €	0,005 €

Table 1: Nett-Costs of different Gateway Numbers [Op08]

2.1.3 SMS-Application Examples

Because it is technically minimalistic, applications using an SMS-Gateway interface tend to be simple. The following list of sample applications is by no means exhaustive, but representative.

2.1.3.1 Google Calendar

As part of the Calendar web-application, Google offers SMS notification of events stored in the online-calendar. This takes the form of 'reminder-SMS', sent to the user before the event takes place. Additionally, several commands can be sent to a dedicated Keynumber of Google prompting calendar information via SMS:

'next': Query the next scheduled event from the Calendar Server

'day': Query all scheduled events for the present day from the Calendar Server

'nday': Query all scheduled events for the next day from the Calendar Server

According to Google, the service is free and available only in the U.S.A. [Go08a]

2.1.3.2 Google Query

In June 2006, Google launched an SMS-Interface for its web-Search application which has since been discontinued. Keywords were sent to a short-code phone number of Google via SMS, and the content of the first Hit was sent back to the querying mobile phone via SMS (in a compacted form). The main idea was to search for retailers and service providers. The cost was .29 €/SMS. Google has however now chosen to grant mobile access to its Search-Engine exclusively through a Java-application. Considering the complexity of the Domain (web-Search), a SMS interface presumably showed itself to be too primitive, and the service was apparently not well received. However, the service still seems to be offered in the U.S.A. [Go08b]

2.1.3.3 'Tamagochi'

This was a short-lived game from D2-Mannesmann, the predecessor of Vodafone in Germany, appearing around 2001. Called 'Zoing', a virtual pet was fed, pampered and educated via SMS commands. Feedback on the state of the pet was also sent back via SMS. In addition to the SMS-interface which was quite expensive (.39 DM/SMS), a Flash-Based web application could also be used to interact with the animal, in this case for free, and with graphical feedback about the pet's situation. At least 10 interactions/day were required to keep the pet alive. The service was discontinued around 2003.

2.1.3.4 Berlin Bus-Schedule

The BVG, the Berlin public transportation authority, supplies current bus- and train schedules to its customers via SMS. Every bus- and train stop has a 6-figure code for every bus/train line that passes through the station. These codes are prominently displayed on the schedule sheets posted at the stops. Sending an SMS with this code to a short-code number triggers a response-SMS from the server with the current next five arrival times at the station. The service is free. [Bv08]

2.1.3.5 Dictionary

'Leo', a popular web-Dictionary, offers an SMS-Gateway interface for translation between English and German. The price for translation is .49€/Word. If further translations for the same word are requested, the price is .19€ for additional meanings, that

can be pulled from the server via the 'more' command. Single letter Option-Codes can be appended (or prefixed) to the word that the user wishes to have translated.

'e': Source word is English

'g': Source word is German

'w': Only common words as response

't': Only technical terms as response

These option codes can be combined. 'Leo' uses a shared Keynumber with the keyword 'leo', indicating that revenues from this service are probably quite low. [Le08]

2.1.4 Customers

Four customer groups can be identified: Party Organizers, who book parties for their events, end-users, who actually play the game, advertisers, that pay to send SMS-messages to the end-users in the database, and advertisers who purchase Ads which are projected onto the 'Statistics-Page' of the Party.

2.1.4.1 Party Organizers

Organizers book parties. They pay a fee, and receive cloth-stickers, tickets, flyers and promotional material via snail-mail. The SMSParty system readies itself to receive SMS messages in the time the party has been booked. In the current version of the system, the organizer does not book the party himself using a web-browser, but does so through a customer representative. Cloth stickers could be 'branded' to support the theme of a party. Tickets are generated by the System as PDF files, and are printed onto perforated A4 sheets. Party Organizers receive individual rates, depending on the previous business relationship with the customer. A very important service to the Organizer are SMS-Mailings, that are sent to previous End-Users of SMSParty who have already attended such a party at least once in a location whose Zip-Code (Postleitzahl) was close to that of the party now booked by the Party Organizer.

2.1.4.2 End Users

End Users are the actual players of the game: They log into the system via the password on their ticket and the 'Login' Command sent to the Server via SMS. Originally the plan was to provide the game free-of-charge to the End User, and only charge the Party Organizer for booking a party. However, market research revealed that Party Organizers would not pay sufficient fees for booking a party, whereas End-Users would not hesitate to pay relatively high fees for sending individual SMS. SMS Gateway service providers enable their customers to exact a fee from End-users sending a message to the gateway, which is billed to the EU through his mobile-phone bill ('Premium Service', see chapter (2.1.1)). It is believed that individual SMS messages to the system could be charged with fees as high as 50c – 1€. Billing End-Users also offers the advantage

of not having to worry about Incoming and Outgoing Message costs. SMS gateways charge for every individual Message received and sent. If the Party Organizer would only pay a lump sum, excessive use of the system by End Users might raise these variable costs up to the point where the party became unprofitable. Charging the End-user eliminates this possibility.

2.1.4.3 Advertisers via SMS

Another possible customer is an advertiser who wishes to promote a product among the End Users that are currently attending or have previously attended a party. Specific groups can be targeted, according to location and theme of attended parties. The invitations sent to End-Users promoting upcoming parties are a special case of this, but other products could also be promoted. However, this product must be sold sparingly, if at all: Every End User has the possibility to opt-out of receiving further promotional messages via the 'Stop' command (see chapter (2.2.3.1)), and selling campaigns to Advertisers might trigger many End Users to do exactly that. It is presumed that advertisements sent out during a specific party for products sold at a specific party would have much higher acceptance rates, but that campaigns promoting a product outside of an ongoing party setting would be ill received. As the database of End User mobile Phone numbers is the most valuable asset of the System, selling to Advertisers via SMS must be carefully evaluated in each case.

2.1.4.4 Advertisers via web-Billboard

Although not foreseen for the upcoming version of SMS-Party, future versions will support a web-Site for each party which shows statistics and events of the ongoing party. Party Organizers are encouraged to project this Page onto a screen at the party location with a beamer. Animations could announce the triggering of a Match, and statistics showing the total number of guests, messages sent etc. could be displayed. At such parties, a further revenue-model is possible: Selling advertising space on this web-Billboard, through films, flash animations, images or 'lower thirds' (i.e. texts scrolling along the lower part of the screen). Advertising via web-Billboard could be booked for a specific location, or for all parties with a specific theme etc. However, acquiring such advertising customers will be slow initially, and managing their accounts and content is technically complex, so this source of revenue remains reserved for future versions of the system.

2.1.4.5 Possible Future Markets

A web-community for End Users is an obvious next step should the product be a success. End Users could log into a community website and track other End Users, send messages to them, browse their party-record, and learn of upcoming events. Building a web-community such as Facebook or MySpace is technically challenging, much more so than the SMS-Party application presented here, and would face established compe-

tion. Outlining the approach is beyond the scope of this thesis, and implementing such a community would only make sense if the unique selling proposition that distinguishes SMS-Party –anonymous matching per SMS- is a success which binds End Users to the brand name.

2.2 Technical Aspects

After the preceding brief overview of the business model of the planned web-application, the following sections explain the technical aspects.

2.2.1 Original Approach

As described in the previous chapter, the application consists of two parts: A booking system, and a game system. Whereas the game system did not change very much from initial planning to implementation, the original concept of the booking system was very different from what was eventually built. In the beginning, the idea was to develop a booking system that would be used by the party organizers themselves: They would open individual accounts, manage their parties, and pay for them via Credit card on a website without the mediation of a customer service. However, in the course of fleshing out the system, I realized that implementing security requirements and billing mechanisms would absorb most of my time, and that I was also not sure how to optimally model the business transactions. Would I have to implement functions for changing the start-time of a party, or would that be a rarely requested feature? What billing methods would prove to be popular, and which might hardly be used? What strategies should I implement to penalize Party Organizers who were late on payments?

At the same time, I also realized that I would need a back-office: Sending out the cloth-stickers that identify the guests of a party, printing out the tickets, and answering Party Organizer questions on the phone would be inevitable. I therefore decided to implement an insecure web-application operated by customer-support representatives in the Back-Office accepting party orders from Party Organizers via phone, fax or mail. This decision enabled me to concentrate on basic functionality associated with the unique selling proposition of my application, and not build a secure, detailed and streamlined web-application. If my business model worked out, I could still transform the web-application designed for customer representatives into a Party Organizer oriented application, with the added benefit of now knowing what business transactions would be the most important and likely to occur, and optimize the web-application for these.

2.2.2 Data Model

In a first Iteration of the Data Model, all In- and Outgoing Messages were persisted in the Database. Eventually, it was seen that Incoming Messages need not be persisted: In case the system crashes, the SMS-Gateway will no longer be able to successfully deliver the incoming messages. When this happens, the Gateway repeatedly tries to redeliver the SMS messages. The responsibility of persisting incoming Messages in case of a system crash is therefore already being shouldered by the SMS-Gateway. The exact protocol of the SMS-Gateway is discussed in chapter (2.2.3.5).

Outgoing Messages do need persistence, however: In case the SMS-Gateway is unavailable, the gaming system repeatedly tries to dispatch the message in certain intervals. Persisting outgoing messages is the job of **SmsMessageQueue**.

The center of the Data Model is the **Party** entity. A **Party** has a unique **Order**, which records the time it was placed by the ordering **Person**, applying a certain **Rate**, i.e. a pricing scheme that may differ from customer to customer, from order to order.

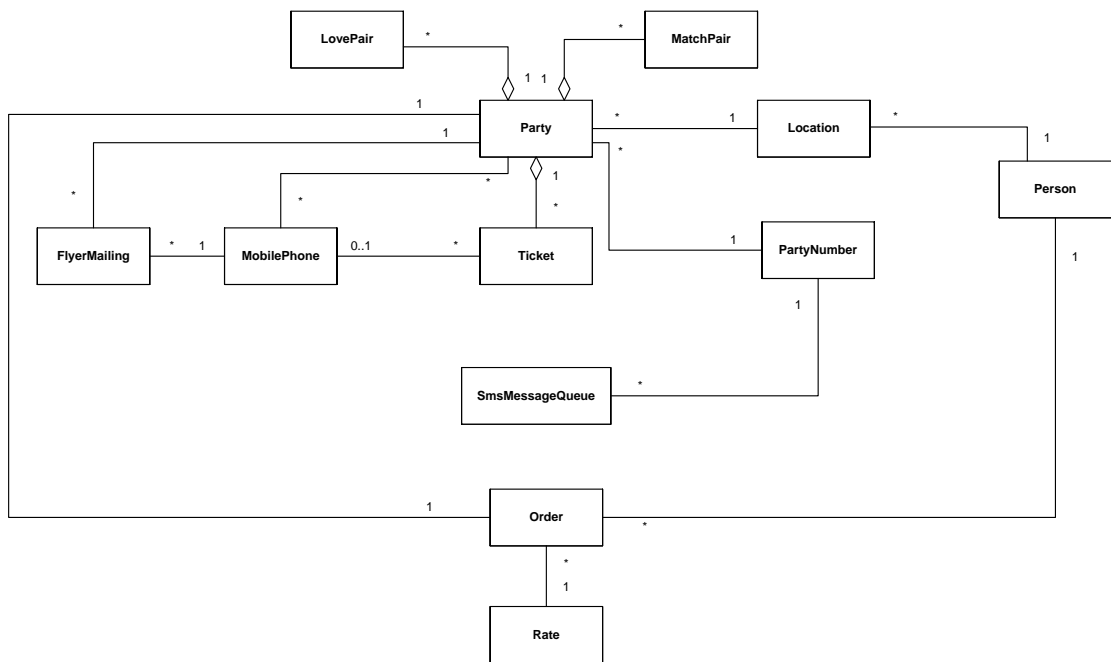


Figure 5: DatabaseModel of the SMS application

A **Person** maintains a set of **Locations**, one of which will be assigned to every **Party** he wishes to organize. Two **Locations** may very well be geographically identical, e.g. if two Party Organizers use the same premises to host their events, but in this case, nonetheless, two distinct **Locations** will be filed in the Database: Contact-Numbers, Location-Style, all these things may differ, even if the geography is the same.

At the creation-time of a new **Party**, a set of **Tickets** is generated. Each set of **Tickets** is also 'physically' incarnated as a PDF Document, destined to be printed out on pre-

perforated sheets of paper and dispatched to the Party Organizer. Each Ticket has a number representing the 'sticker-number' a party guest will be wearing. A corresponding password, also printed onto the ticket, has to be combined with the Ticket-number in order to successfully log into the game via the "Login" command (see chapter (2.2.3.1)).

If a party guest logs into the system via a successful "Login" command, he is identified by his GSM-number. Now there are two possibilities: Either the guest has never visited a previous **Party**, in which case a new **MobilePhone** entry is created for him, or the guest has already been a participant in at least one previous **Party** in which case his **MobilePhone** entry is retrieved from the Database. In either case, the **Ticket** is devalidated (this is done by setting a Boolean Flag in the Table-Entry), and associated with the **MobilePhone**. Also, in both cases, a direct relationship between **MobilePhone** and **Party** is established: **Tickets** of a **Party** will be discarded after an event is over. However, for promotional purposes, the information about which **MobilePhone** has been a player in which **Party** should be retained, information that would be lost if only the **MobilePhone-Ticket-Party** connection existed, when the **Tickets** associated with the **Party** are removed from the system when the event is over.

This brings us to **FlyerMailing**: When a new **Party** is created in a certain **Location**, the Database retrieves all **MobilePhones** which have previously been guests at a **Party** at a **Location** near² the one of the new **Party**. These **MobilePhones** are then earmarked for promotional SMS-Messages advertising the new **Party** via **FlyerMailings**. A **FlyerMailing** entry is therefore a commitment from the system to send out a SMS to a **MobilePhone** promoting a specific **Party**. The SMS are dispatched 48 hours before the beginning of the **Party**.

The Messaging System is described in more detail in chapter (2.2.3). As already mentioned, Incoming messages are not persisted, but lead to the creation of volatile Message-Objects which are immediately processed and change the state of the Database. 'Love' Messages generate an entry in the **LovePair** table, persisting which tag-number is interested in which other tag number. In case of mutual interest, a **MatchPair** entity is generated, and triggers the dispatch of a SMS. All outgoing SMS are persisted in **SMSMessageQueue**. There is no connection between **MobilePhone** and **SMSMessageQueue**, because not all outgoing SMS-Messages will be addressed to registered **MobilePhones**: In case of a faulty Login, or an ingame-message belonging to an unregistered mobile-phone, outgoing messages are dispatched to mobile-phones that are not in **MobilePhone**. There is no connection between **Party** and **SMSMessageQueue** because, in this case, **SMSMessageQueue** would not be able to find the GSM-number

² A 'nearness' function can be as simple as finding those ZipCodes that start with the same digit (for Germany), or as complex as determining geographical coordinates for each address, evaluating the distance, and returning all addresses no farther than a certain number of kilometers away. For this application, the ZipCode approach was used.

of the addressee in **MobilePhone** anyways, and it is therefore necessary to store the destination-GSM-number in **SMSMessageQueue**, and a connection to **MobilePhone** becomes superfluous.

The last table to be discussed is **PartyNumber**. It is here that the Outgoing SMS Gateway URL is specified, and also the GSM-Number which should be displayed as dispatcher. Every **Party** has one **PartyNumber**, as does every **SMSMessageQueue**.

2.2.3 Messaging System

The following chapters deal with the input and output of text messages to and from the SMS-application.

2.2.3.1 Commands

In addition to the 'LOVE' command mentioned in chapter (1), several other commands can be sent to the central number by users. An overview of all incoming messages is given in Table 2.

Syntax	Example	Semantic
Login <Password> [<Nickname>]	Login 01122!32xzes Bob	Validate Guest and assign name
Love {<tag>}	Love 12 34 56	Register interest of sender in tagged guests
Hate {<tag>}	Hate 12 24 56	Ignore interest of sender in tagged guests
Freeze	Freeze	Refrain from matching
Unfreeze	Unfreeze	Restart matching
Status	Status	Inform sender of party statistics
Stop	Stop	Remove sender from mailing list
SyntaxError	Luv \$12	Inform sender of malformed message

Table 2: Incoming Messages ('Commands') of the SMS Party Application

The 'Login' message, compulsory for the user to start playing, validates the user by verifying the password supplied to the guest through a ticket. A nickname can optionally be entered.

'Hate' removes a formerly 'loved' guest from the user's list of love-interests.

'Freeze' suspends all matching for this user. This might be called for immediately after a match: Another match might not be wished for when a successful one has taken place. However:

'Unfreeze' once again activates the list of candidates submitted through the 'Love' command, if it has been deactivated by 'Freeze'.

'Status' is a command that triggers a Status message from the server: Statistics about the ongoing Game are transmitted to the user: How many (but not which!) guests have put him on their love-lists, and also on how many love-lists the average user is.

'Stop' removes the user from a mailing list promoting future parties/events. The list of mobile-phone numbers collected during games is a valuable asset, but the user can opt-out of receiving commercials of this kind.

'SyntaxError' is not a command in itself, but rather a catch-all for malformed commands. It triggers an outgoing message informing the sender of his or her mistake.

2.2.3.2 Responses

The Commands listed in (Table 2) trigger one or more outgoing Messages, called "Responses", generated by the web-application. A complete list can be seen in (Table 3). Most of the Responses are self-explanatory.

'Status Out' is a response triggered by the 'Status' Command. It informs the guest how many men, on average, are interested in each woman, and how many women, on average, are interested in each man. It also tells the guest how many other guests are interested in him/her, i.e. have put him or her on their Love-List. Of course, only the absolute number, not the identity of these courtiers is given. The 'Status Out' Message thereby allows the guest to evaluate his or her attractiveness: If he or she has more courtiers than the average male, respectively female, then their attractiveness can be presumed to be greater.

'Invitation' and 'Cancellation' are Outgoing Messages that are not really 'Responses', as they are triggered by the System, not by Incoming Messages: Each Party will invite guests to its event via 'Invitation'-Outgoing Messages sent to guests who have already attended a party in an area near the one of the upcoming event. This happens 48 hours before the beginning of the party. The text of this invitation is tailored to the wishes of a Party Organizer ("Flyer-Text"). However, the last part of the Message is reserved for an "unsubscribe-blurb", telling the prospective guests how to stop receiving messages of this kind through the 'Stop' command.

The 'Cancellation' Outgoing Message is needed in case the party is cancelled after 'Invitations' have already been sent out. The content of this message ("Dier-Text", rhymes with "Flyer-Text") is also up to the Party Organizer, the last part again being reserved for the "unsubscribe-blurb".

Name	Syntax	Example	Semantic
Login Confirm	[<Nickname>,<tag>], you have successfully logged into the Party "<Party Name>".	„Bob (32), you have successfully logged into the Party „Siesta Mexicana“.	Confirm valid Login.
Login Fail	Login failed. Please try again.	Login failed. Please try again.	Inform about failed Login.
Love Confirm	Guests {<tag>} have been added to your Love-List.	Guests 34 55 89 have been added to your Love-List.	Confirm additions to Love-List.
Love Fail	One or more guests could not be added to your Love-List. Please try again.	One or more guests could not be added to your Love-List. Please try again.	Inform about failed Love-Command.
Hate Confirm	Guests {<tag>} have been removed from your Love-List.	Guest 32 has been removed from your Love-List.	Confirm deletions from Love-List.
Hate Fail	One or more guests could not be removed from your Love-List. Please try again.	One or more guests could not be removed from your Love-List. Please try again.	Inform about failed Hate-Command.
Match	You have a Match! Guest <tag>[(<Nickname>)] has also expressed interest in you!	You have a Match! Guest 23 (Alice) has also expressed interest in you!	Inform about Match.
Freeze Confirm	Your Love-List has been frozen, and you will no longer be matched. Send 'unfreeze' to continue game.	Your Love-List has been frozen, and you will no longer be matched. Send 'unfreeze' to continue game.	Confirm successful Freeze Command.
Unfreeze Confirm	Your Love-List has been reactivated, and you will again be matched.	Your Love-List has been reactivated, and you will again be matched.	Confirm successful Unfreeze Command.
Status Out	Men->Women Lovers: ~<value>, Women->Men Lovers: ~<value>, You: <value> Lovers	Men->Women Lovers: ~8.4, Women->Men Lovers: ~5.2, You: 12 Lovers	Inform sender of current statistics
Invitation	<Flyer-Text> Send „Stop“ to Cancel SMS-Matchmaker	Party im „Vista-Club“, Begin 20:00 Send „Stop“ to Cancel SMS-Matchmaker.	Inform a potential guest of an event, and how to unsubscribe.
Cancellation	<Dier-Text> Send „Stop“ to Cancel SMS-Matchmaker	Party at the Vista-has been cancelled due to bad weather. Sorry!! Send „Stop“ to Cancel SMS-Matchmaker.	Inform a potential guest of a cancelled event, and how to unsubscribe.
Syntax Error	We could not process your message. Please try again.	We could not process your message. Please try again.	Inform sender of malformed command.

Table 3: Outgoing Messages ('Responses') of the SMS Party Application

2.2.3.3 Message Processing

Messages change the state of the system, but are not persisted as such. The actual SMS Texts of Outgoing Messages are queued into the Database in mere text form, but these **SMSMessageQueues** are artifacts of volatile (outgoing) Message objects, not Messages in the actual sense, but commands to text certain Strings via the SMS Gateway to certain GSM-numbers. Messages in the sense of this chapter are volatile Objects created when parsing an incoming SMS-Message (**SmsInMessage**), or triggered through the business logic of Incoming Message Objects (**SmsOutMessage**).

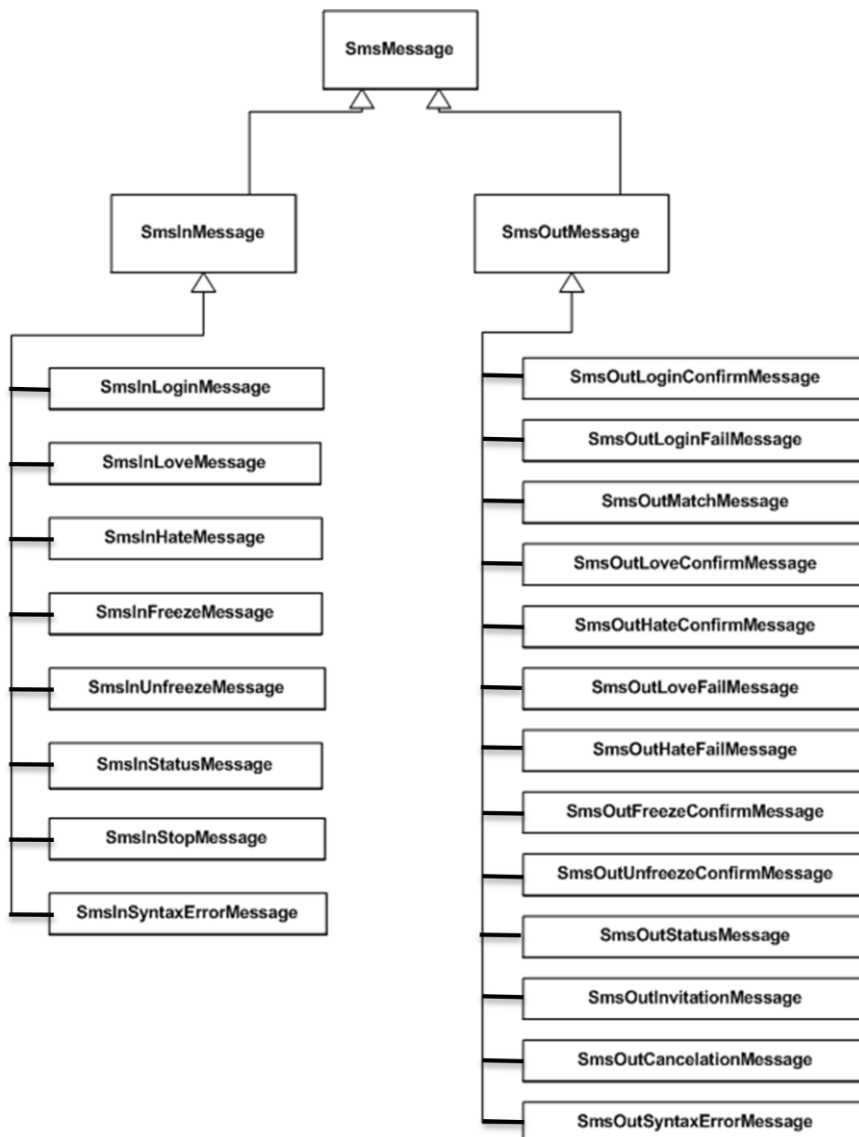


Figure 6: Hierarchy of the **SMSMessage** Class

The Messaging mechanism from the User-sent SMS Message to the User-received SMS Message is explained in the Activity Diagram Figure 7).

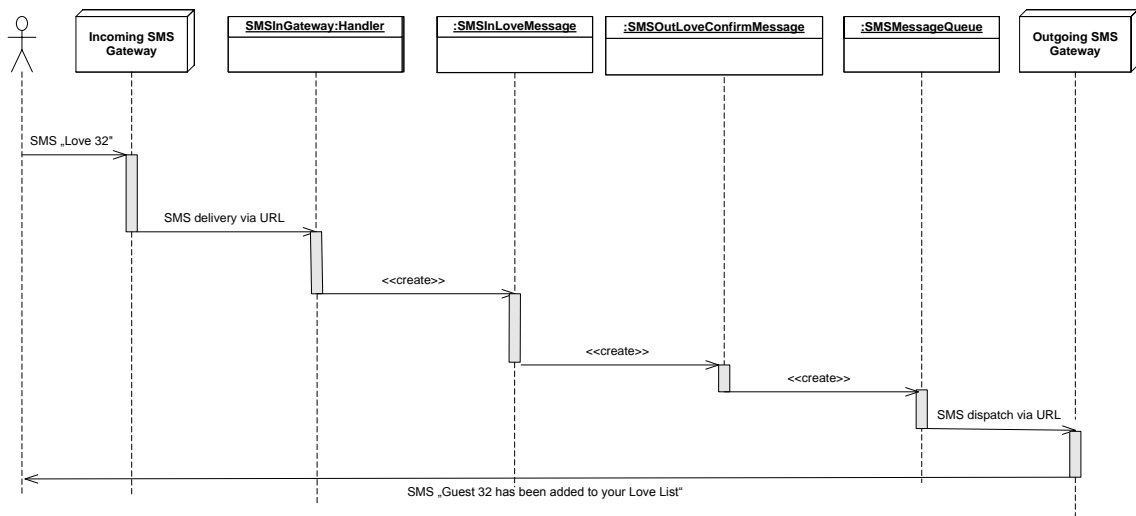


Figure 7: Processing a Message

In a first step, the guest (shown as a Stick-Figure) sends a SMS via his mobile phone. This message is received by the Incoming SMS-Gateway, which is supplied by an external vendor. The Incoming SMS Gateway will then ‘touch’ a URL, which we have to communicate to the external vendor operating the gateway, which will be handled by the **SMSInGateway** Handler. This is a Toro Handler which interprets the parameters transmitted in the URL. The mechanism of the Incoming SMS Gateway differs slightly from vendor to vendor, but is described for the Click-A-Tell vendor in chapter (2.2.3.5).

The **SMSInGateway** Handler parses the SMS message and creates an instance of the appropriate Message Class. In our example, an **SMSInLoveMessage**. This object is then responsible for changing the state of the System, and thereby triggers the creation of **SMSOutMessages**. In our case, the logic of **SMSInLoveMessage** would create a **SMSOutLoveConfirmMessage** (as shown), add the listed tag-numbers to the love-list, then check for matches, and – in case a match has occurred - create the two appropriate **SMSOutMatchMessages** (not shown). The generated Message **SMSOutLoveConfirmMessage** then generates the text that should be dispatched via SMS and instantiates it in the form of **SMSMessageQueue** objects. In regular intervals, the objects of **SMSMessageQueue** are dispatched via the Outgoing SMS Gateway supplied by a vendor. This is also a URL which expects certain parameters and then dispatches the SMS via GSM to the mobile phone. The mechanism of the Outgoing SMS Gateway is similar to the Incoming SMS Gateway, and described in chapter (2.2.3.4), for the specific vendor Click-A-Tell.

The Logic controlling the Commands and the Responses is straightforward. For the Login and Love commands, the Business Logic is shown in the following activity diagram:

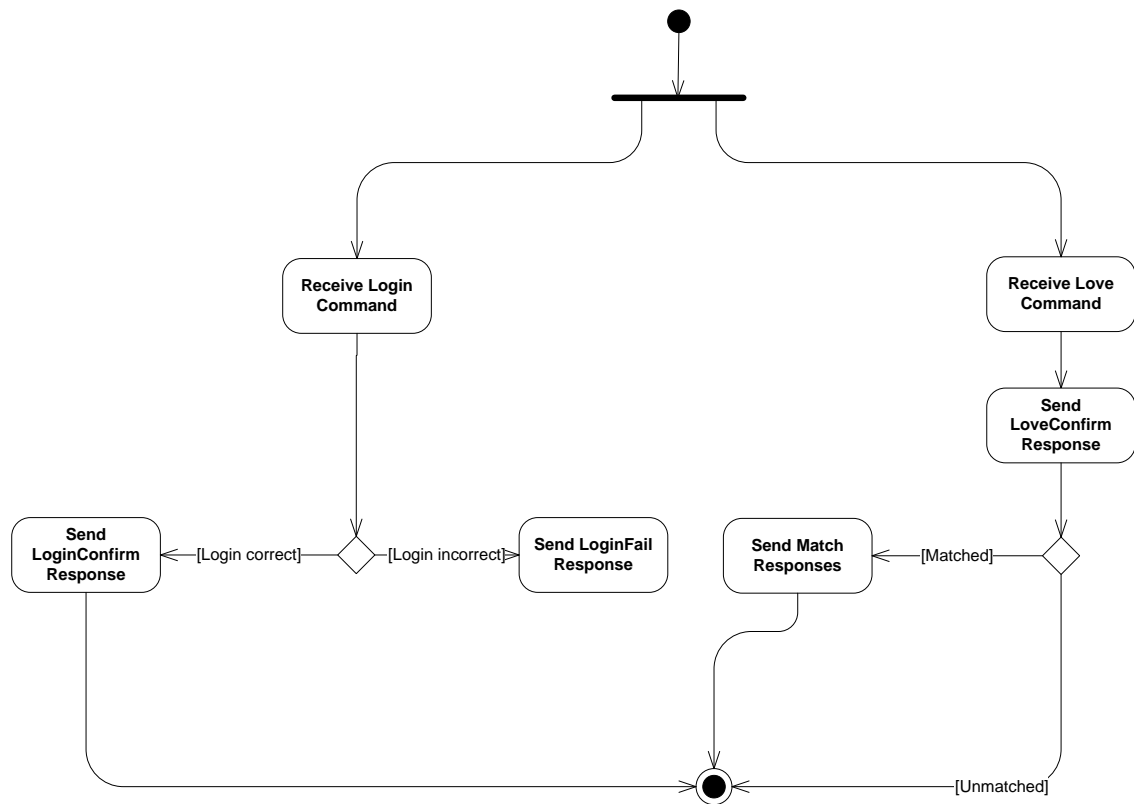


Figure 8: Activity Diagram of 'Login' and 'Love' Commands

2.2.3.4 Outgoing Messaging Gateway

The interface between the gaming-application running on a server and the GSM network connected to the mobile phones and capable of wirelessly dispatching the messages is a URL address, supplied by the Gateway vendor. This URL expects several parameters that define the content of the text message, addressee (=mobile-phone) and assure sender-authenticity.

The sequence and nature of these parameters differ from vendor to vendor. As an example, the syntax of a typical vendor, the South-African company Clickatell will be described here [CI08a].

Before being able to send out SMS, an account must be acquired from the vendor. A username, password, contact details and a payment method must be provided. Once this has been done, SMS can be triggered by touching a URL formed in the following way:

```
http://api.clickatell.com/http/sendmsg?api_id=xxxx&user=xxxx&password=xxxx&to=xxx  
x&text=xxxx
```

where:

api_id: An account number supplied by the vendor.

user: The login of the account holder.

password: The password of the account holder.

to: The destination GSM number. No '00' or '+' prefix should appear in the front of the (mandatory) country code

text: The payload of the SMS. The actual text. The text must be no longer than 160 characters.

Internet access from within a Java program via the HTTP protocol can implemented using the **java.net.HttpURLConnection** and the **java.net.URL** classes.

First, a URL object must be instantiated with the appropriate parameters, e.g.:

```
URL url =new URL ("http://api.clickatell.com/http/sendmsg?api_id=12345&user=shenckel  
&password=top-secret&to=491773061203&text=This is a test.");
```

Then, this URL can be touched (and, in our case, the message sent), via:

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

2.2.3.5 Incoming Messaging Gateway

Sending SMS is one part of the application, receiving Mobile Originated SMS the other. The mechanism of receiving incoming SMS with the SMS-application is realized by registering a callback-URL with the SMS-Gateway provider. Each time a SMS is sent to the number registered with the SMS-Gateway provider ("Party-Number"), this URL is touched.

The details differ from vendor to vendor. The example described below once again follows the conventions of Clickatel [CI08b]:

The parameters passed via the GET method in the URL are similar to those for Outgoing Messages:

Suppose we have registered the domain name www.sms-dating-game.de, and have an instance of our Toro SMS-Dating Game application running on a server under this name. If we register the callback-URL www.sms-dating-game.de/SMSInGateway.htm with the Gateway Provider, and a SMS is sent to our Party-Number, the Gateway Provider will touch the following URL:

```
http:// www.sms-dating-game.de/SMSInGateway.htm?api_id=xxx&from=handset_number
&to=party_number&timestamp=2005-01-06+12:32:10&text=xxx&charset=ISO-8859-1&udh=
```

The parameters passed via HTTP in the URL are similar to those for Outgoing Messages:

api_id:	An account number supplied by the vendor.
from:	The mobile Phone Number of the sender. No '00' or '+' prefix will appear before the country code.
to:	The Telephone-number the SMS was sent to.
timestamp:	The date and time the message was received, in MySQL format.
text:	The payload of the SMS. The actual text.
charset:	The character set of the text. Can be ignored.
udh:	Header data. Can be ignored.

By a Toro naming law (see chapter (3.10), accessing this URL will call a Toro Handler named **SMSInGatewayHandler.java**. It is the job of this handler to interpret these parameters.

3 Tutorial

We now come to the main part of the thesis – a tutorial explaining how to build a web-application with the Toro-framework.

3.1 Goal

The aim of this tutorial is to learn how to build a small web-application using the Toro webvisualization and Persistence framework. Recently, for my thesis project, I built a web-application in Toro allowing Party Organizers to purchase a Mobile-Phone game for their event: The game rules are simple: Guests log into a computer via their mobile phone and a password they are given, and can then send Short messages to this server, stating which of the other party guests they find attractive. In case of a match – two guests finding each other mutually attractive – both guests are informed. We will implement a part of this web-application. You (the organizer) have an account that you log into. If you do not have an account, you can set one up. Once you are logged in, you can purchase new parties, or cancel existing ones. The data model is simple:

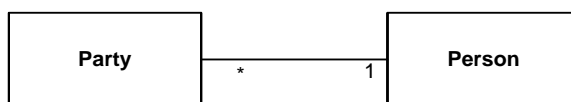


Figure 9: Data-Model of Toro-Tutorial

When developing a web-application, I start out with the data model. Having done that, I make a list of things I want to do with that data through my web-application.

In this case, I want to:

- Log into an existing account
- Create a new account
- Edit account data
- View parties
- Create new parties
- Delete parties

When I know what to do, I draw a state diagram with each state representing an html page, and the arrows between states representing the links between the pages. My web-application then looked like this:

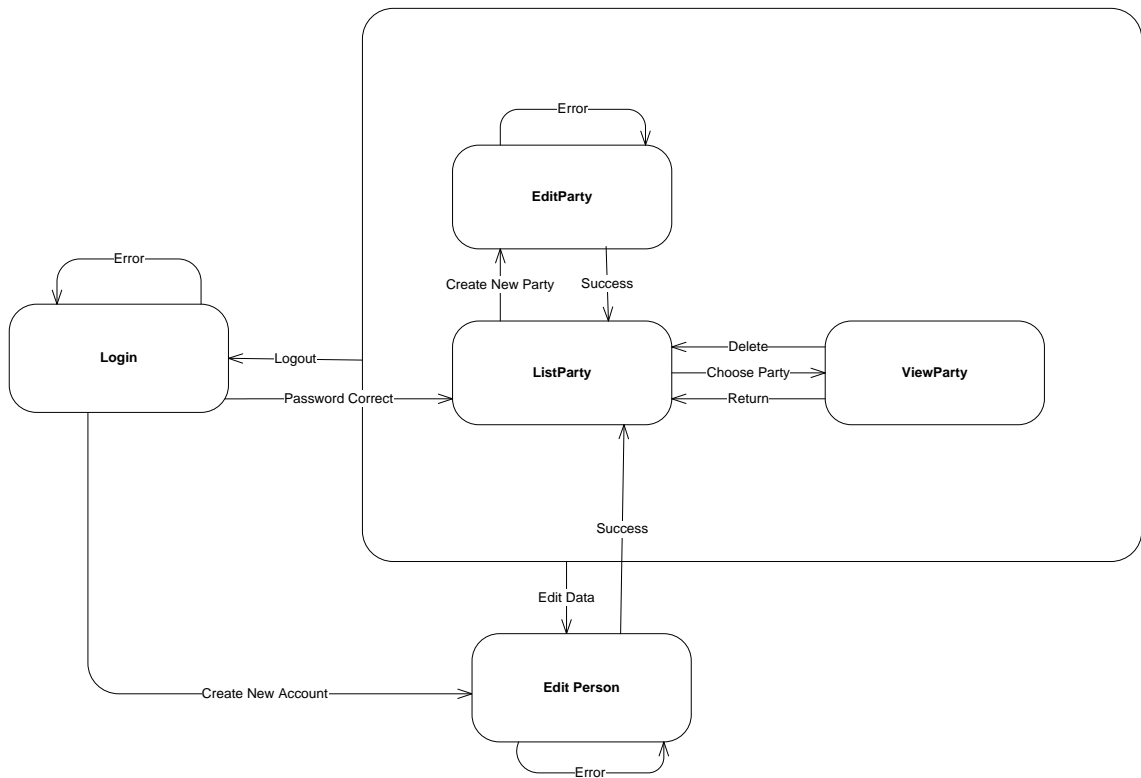


Figure 10: State Diagram - User Interface of Tutorial

Let us now install Toro onto the PC you will be using.

3.2 Installation

Toro depends on a Database application and the Eclipse Integrated Development environment (IDE). If you have not already done so, install a Database of your choice (the tutorial will presume MySQL). For this tutorial, we will require a Database called 'minisms', with a user called 'root' and a password 'mYsql'. You can create the Database by logging into your console and executing:

```
CREATE DATABASE minisms COLLATE utf8_bin
```

3.2.1 Installation – Basic

Toro is distributed in one Zip-File containing Eclipse, Toro and all required plugins, settings and projects.

Download the Toro Zip **development.zip** from the Toro-website.

All you have to do is to extract the content of this zip-file to a folder called `/development/` located in the root directory or a folder of your choice. Skip to chapter (3.3).

3.2.2 Installation - Expert

Installing Eclipse and Toro:

Toro requires at least Eclipse Version 3.3.* or above. The following installation assumes Eclipse Version 3.4 (Ganymede).

Download the Eclipse Zip File from <http://download.eclipse.org/eclipse/downloads/>

Unzip the eclipse Directory inside the Zip-File to a location of your choice on your hard drive:

```
[your path]/eclipse
```

Download the Toro Zip **toro.zip** file from the Toro-website. Unzip the Toro Directory inside the Zip File to the same location on your hard drive you chose for the eclipse Directory:

```
[your path]/toro
```

Create a new Workspace Directory. A good practice is to put it in the same location on your hard drive you chose for the eclipse and Toro Directories:

```
[your path]/eclipse-workspace
```

You have now created all the directories needed for your installation. The Toro framework needs to know three things:

- Where you have installed Eclipse
- Where you have placed your Eclipse Workspace
- Where you have installed the Java Virtual Machine

This information must be written into the file:

```
[your path]/toro/localProperties.txt
```

This file, as the name suggests, stores all local settings of your machine for Toro.

Opening **localProperties.txt** displays the following four variables:

```
eclipse.dir=D:/demo/toro/essay/eclipse
eclipse.workspace.dir=D:/demo/Toro/essay/eclipse-workspace
JAVA_HOME=D:/Programme/Java/jdk1.6.0_02/bin
PERFORCE_CLIENTSPEC=th.buechner_SRVMATTHESV11_main
```

Replace this text appropriately:

```
eclipse.dir=[your path]/eclipse
```

```
eclipse.workspace.dir=[your path]/eclipse-workspace  
JAVA_HOME=[full path name of your Java Virtual Machine]  
PERFORCE_CLIENTSPEC=th.buechner_SRMATTHESV11_main
```

The last line is only needed if you have Perforce, a proprietary revision control system, installed on your computer, and a corresponding account with which to keep your Toro installation current. This case will not be assumed for the tutorial, and you can therefore simply delete the last line.

You must now start Eclipse to continue the installation. You should *not* start eclipse directly by launching [your path]/eclipse/eclipse.exe. For Toro to work correctly, **eclipse.exe** must be called with a list of specific command line arguments. Instead, launch:

```
[your path]/toro/start-eclipse.bat
```

This .bat file assures that **eclipse.exe** is called with the appropriate command line arguments.

You might want to create a more readily accessible link to this **start-eclipse.bat** file for future use.

Toro depends on certain frameworks from third parties, specifically, the GEF (Graphical Editing framework). Installing this framework can be done from inside of Eclipse:

Go to Help->Software Updates..., and choose the 'Available Software' tab. Then type 'gef' into the search-bar. Select the Checkbox for the Graphical Editing framework, click install, accept the license agreement, then let Eclipse restart.

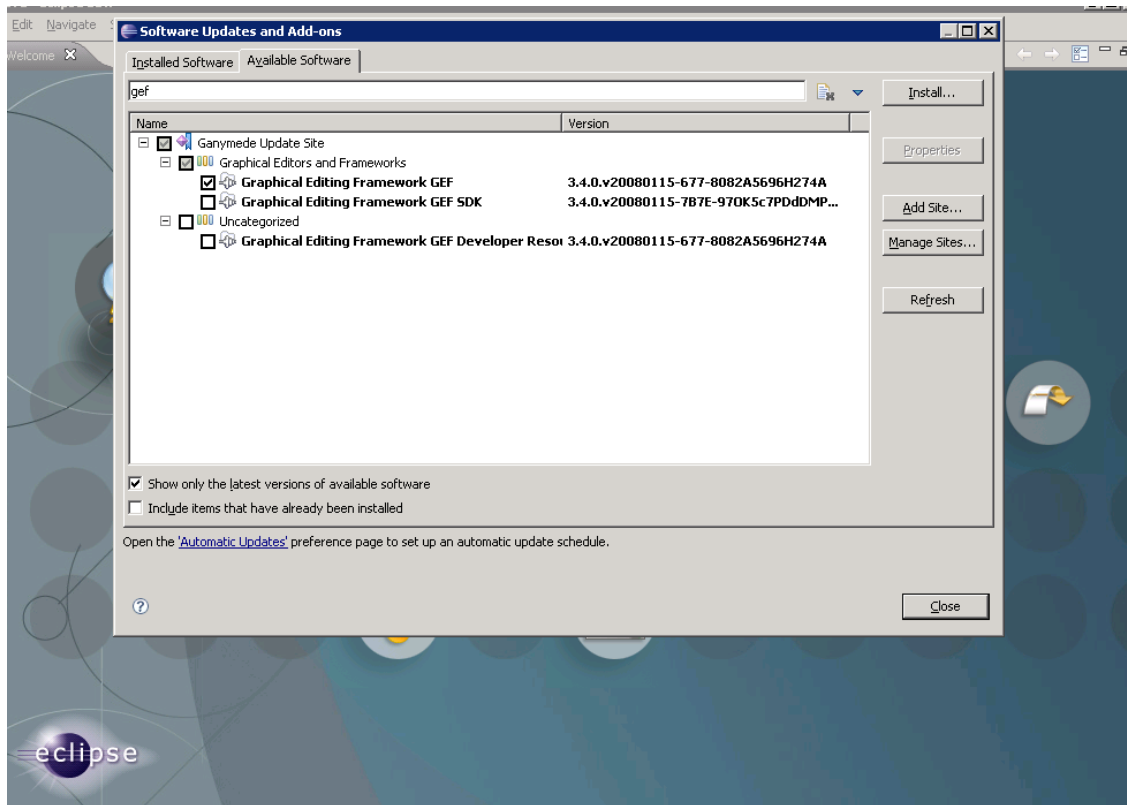


Figure 11: Installing the GEF

Once Eclipse has restarted, we can install the Toro plugins. This is done by telling Eclipse where to look for Toro plugins through a so called 'extension file' in the Toro directory. In Eclipse Ganymede, registering extension files is no longer a standard feature, and you must manually activate this capability:

TIP: In chapter (3.4), there is a tip on how to import Toro-friendly preferences into Eclipse (see Figure 21). You might want to do this here, and can then skip activating the 'extension files' capability, which is part of the Toro-preferences you import there. However, importing the preferences will also change your syntax-highlighting, something you might not wish to do. If that is the case, continue with the next step, otherwise install the preferences as described, and go immediately to the step shown in Figure 13).

Select Window->Preferences, and then choose General->Capabilities. Click on the Checkbox next to 'Classic Update', and click 'OK'.

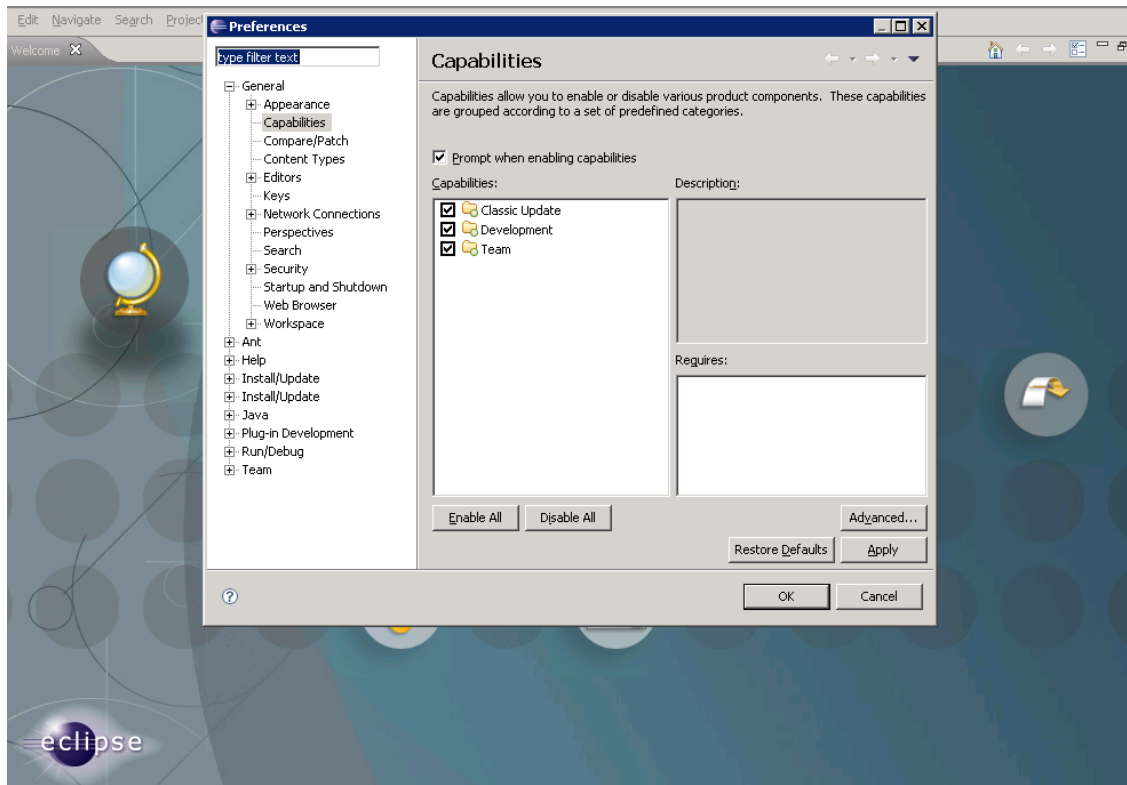


Figure 12: Reactivating Extension Locations in Ganymede

We can now register the extension location:

Choose Help->Software Updates->Manage Configuration..., and then click on the 'Add an extension location' Link.

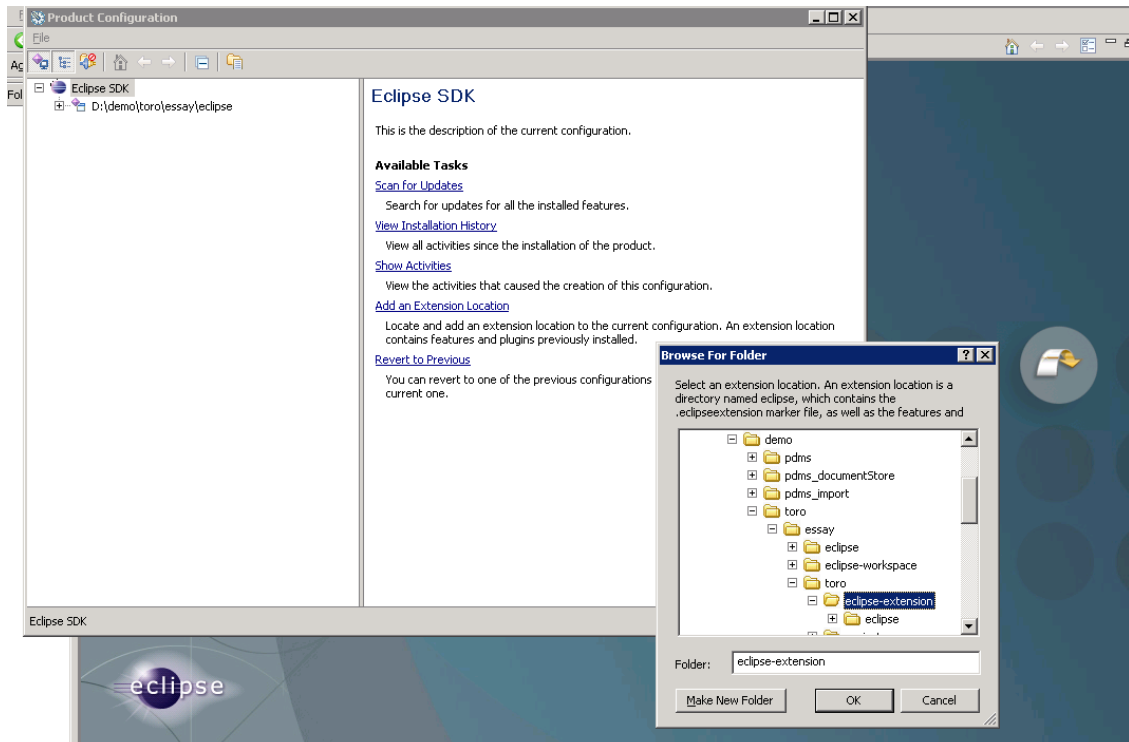


Figure 13: Registering the Toro Extension Location

In the directory selector that appears, select the following directory:

`[your path]/toro/eclipse extension/`

Confirm with 'OK', and allow Eclipse to restart.

Finally, we must import two projects into our Workspace. One, called **miminal123**, is a Toro prototype application which will be our starting point for this tutorial. The other project, **platform-toro**, is required for all Toro projects: Toro is a **'framework'**, which means that the actual **main()** method of your application will not be supplied by your program, but by the **platform-toro** project, which is the **'frame'** in which your program **'works'**.

To import these two projects, select:

File->Import and then General->Existing Projects into Workspace

In the 'Select root directory' box, browse to the Directory:

`[your path]/toro/projects/`

In the selection box, a whole list of projects will appear. Uncheck all, **except minimal123** and **platform-toro**, and import them by clicking on 'Finish'.

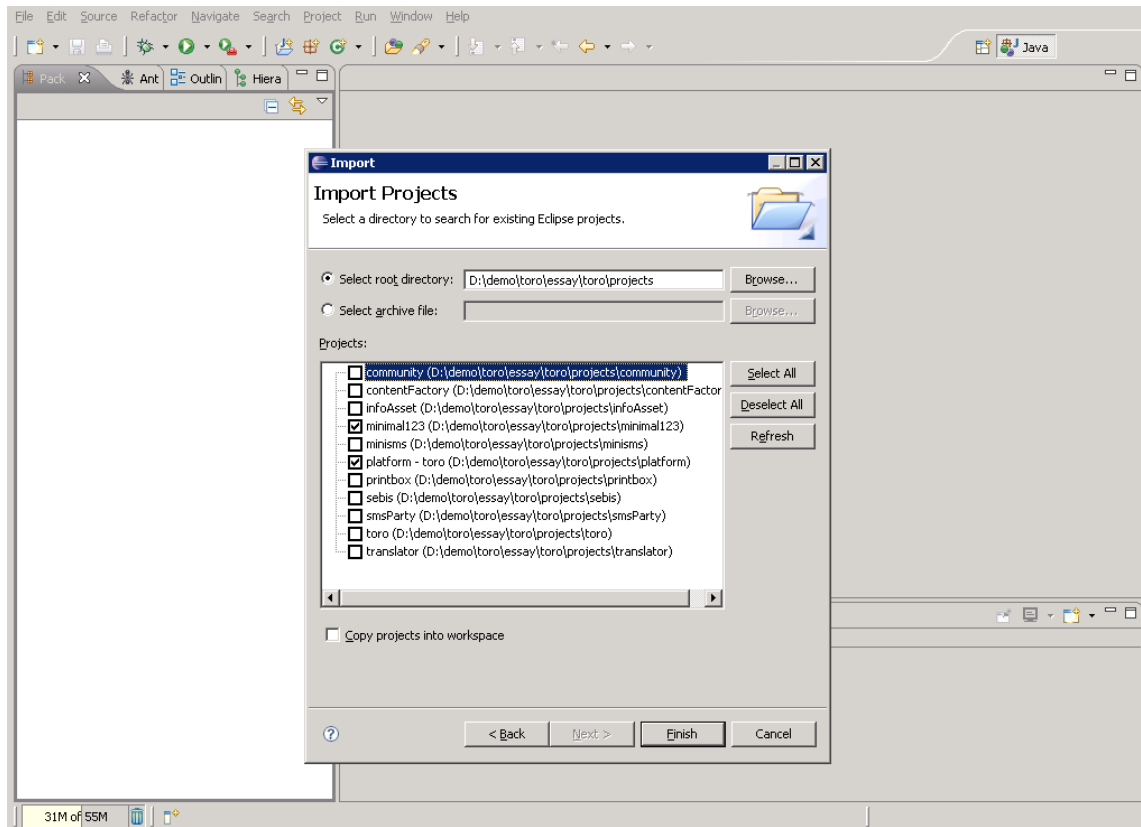


Figure 14: Importing minimal123 and platform-toro

We are now ready to create our own tutorial project.

3.3 Instantiating a new project

A good starting point for a new Toro project is the 'minimal123'-project that comes with your Toro installation. This application is a very basic application, and looking through its structure will teach you much about Toro, so feel free to do so. To build a new application, you make a copy of 'minimal123' under another name. There is an Ant-Script in 'platform-Toro' which copies 'minimal123', renaming it to a name of your choice. Let us do this: In the 'Ant-View' tab (in case it doesn't appear, you have to open it with 'Window->Show View->Ant'), click the 'add Build Files' icon, and select 'build.xml' from the 'platform-toro' folder:

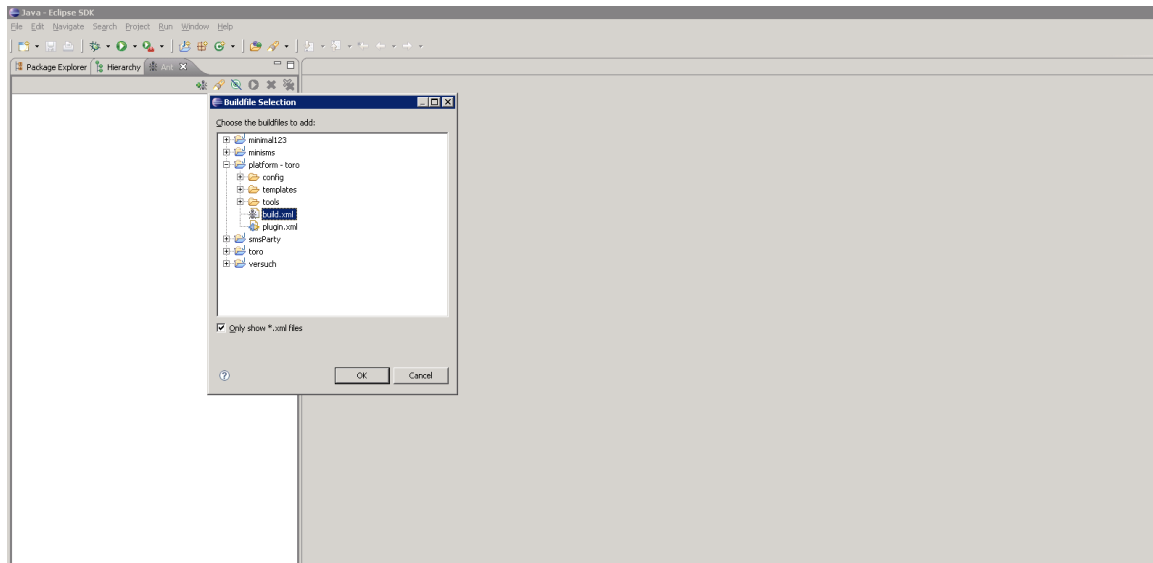


Figure 15: Adding the Ant Build File

There are several commands (‘targets’) in the Build File. Double click on ‘new project’:

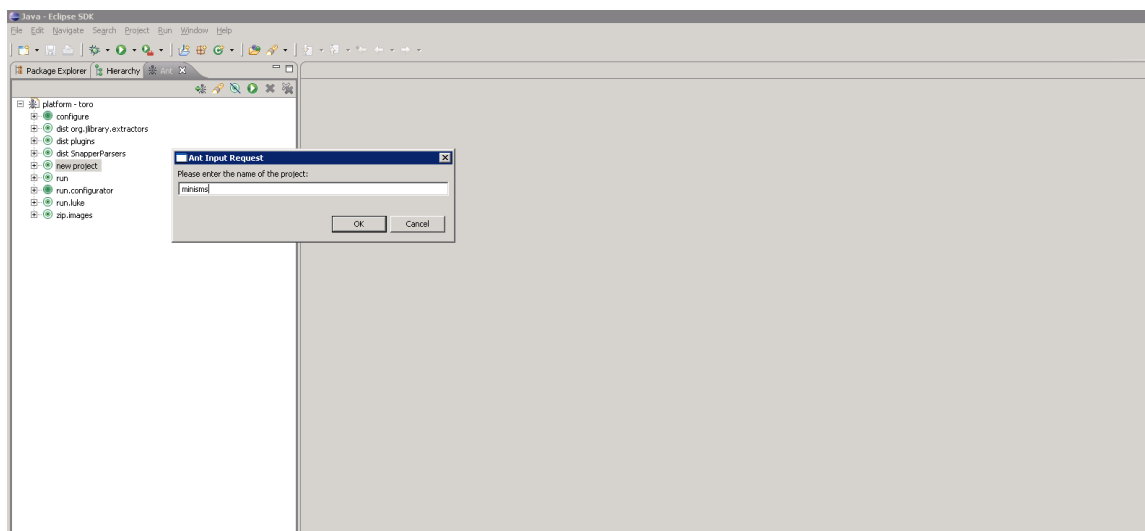


Figure 16: Executing the ,new-project‘ script.

The prompt of the ant-script asks for the name of the new project. We will call it:

‘minisms’

We still have to import our copied project into the workspace. Choose ‘File->Import->Existing Project into Workspace’, and then browse to the `\toro\projects` folder. Deselect all projects except ‘minisms’, and import. You should now see the new project in the package explorer.

How do we run our new project? In the ‘minisms’ project folder in the project-explorer, you will find a file called `minisms-run.launch`. Right-Click on this file and choose

'Run-As->minisms-run'. This `minisms-run.launch` file contains all the settings needed to start the application.

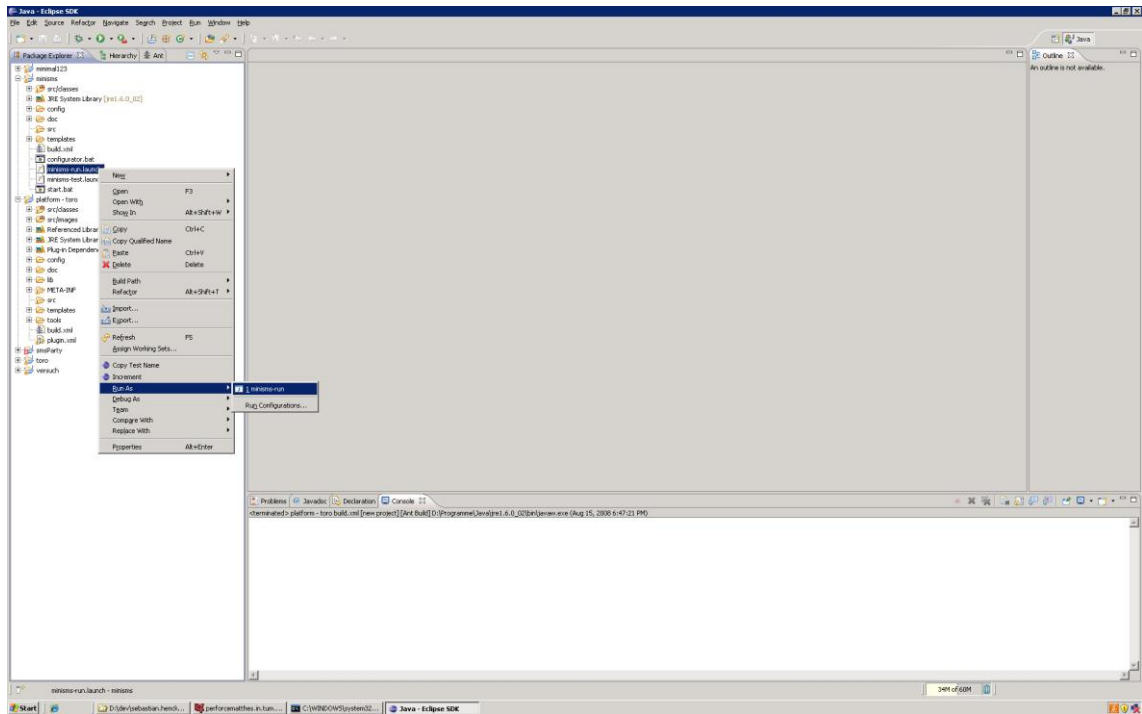


Figure 17: Launching minisms

Before the application launches, Toro prompts you whether it is okay to flush the data in the Database. You must enter 'Y' in the console (make sure it is a big 'Y'). Next, open the browser of your choice, and point it to <http://localhost:8083/>. The following website should appear:

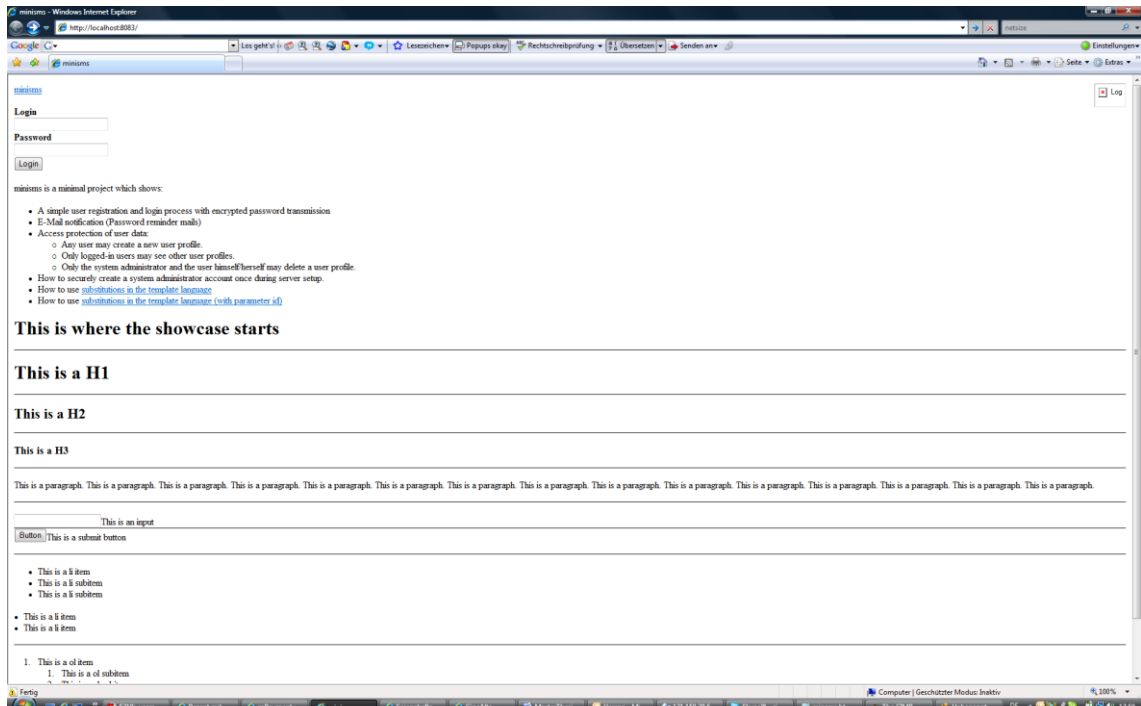


Figure 18: The opening screen of minisms

By default, the port of the webserver is set to 8083. If you do not enter a web-page, the default page displayed is 'home.htm'. For our new project, this is a page inherited from 'minimal123', and is a small showcase for Toro. We will now begin to change our new application to suit our needs.

3.4 Implementing Persistence – Assets and Properties

In our project, we have two persistent classes, **Person**, and **Party** (see Figure (9)).

A **Party**, in our case, is a social event, which should be described by the following attributes:

Party
+name: String (maximum size = 30) +location: String (maximum size = 30) +size: Integer +price: Integer +begin: Date +purchaseDate: Date +rateCategory: DomainValue

Figure 19: Structure of the Party Asset

Toro supplies us with a class called **BaseAsset**, ('Asset', for short) from which we derive the persistent Classes we wish to implement. All of the Assets we create must be

placed in the 'assets' package of the project (`de.infoasset.minisms.assets`). Let us create a class file called **Party**, derived from **BaseAsset** in the 'assets' package:

```
public class Party extends BaseAsset {

    /* Define Properties and Roles (Associations to other Assets) here */

}
```

A word of caution: 'import' statements at the beginning of a listing are omitted for clarity in this tutorial. Appendix (A) lists all files of the tutorials with their import statements. In order to compile the example, the correct import statements must of course be supplied.

TIP: Eclipse helps you import the correct classes if you haven't already done so: If a class you wish to use (such as **BaseAsset**) has not been imported yet, the IDE will underline it in red. You can then 'quickfix' this problem by clicking the Error-Decorator on the left margin of your Edit-Window and choosing 'Import [Missing Class Name]' Sometimes Toro will use class names that also exist in other Java Packages. Be sure to import the right ones.

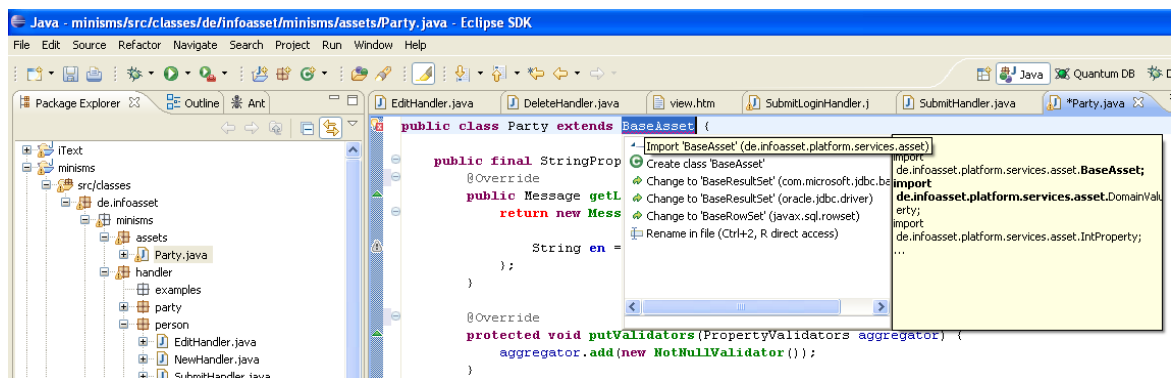


Figure 20: Quickfixing a missing import statement.

Toro-Objects of this type can be persisted in the database without JDBC-like overhead, searched for through a simple query-language, and are easily displayed on webpages.

At this time, we have not yet defined any attributes of our Persistent Asset Class **Party**. Attributes are called 'Properties' in Toro.

Let's start with the 'name' Property. In order to establish a field of type 'String' with attribute name 'name', we write the following code into the **Party** Class:

```
public class Party extends BaseAsset {

    public final StringProperty name = new StringProperty() {

        /* Define Characteristics of StringProperty here */

    }

}
```

```
} ;  
}
```

The reason a property is declared 'final' lies in Toro itself: Once a property reference is assigned, the framework depends on it no longer changing. So be polite, and make your Properties final.

As you can see, the **StringProperty** 'name' is declared via an Anonymous Inner Class (AIC). This is very typical for Toro. (See the Excursion on AICs below)

TIP: As part of the Toro installation package, there is a set of templates that can make your life easier when typing in the definitions of typical Toro objects such as Properties and Messages. Before you can use these templates, you must import them into your Eclipse-IDE. This is done by choosing File->Import->Preferences, and then choosing the `./toro/preferences.epf` Preference file:

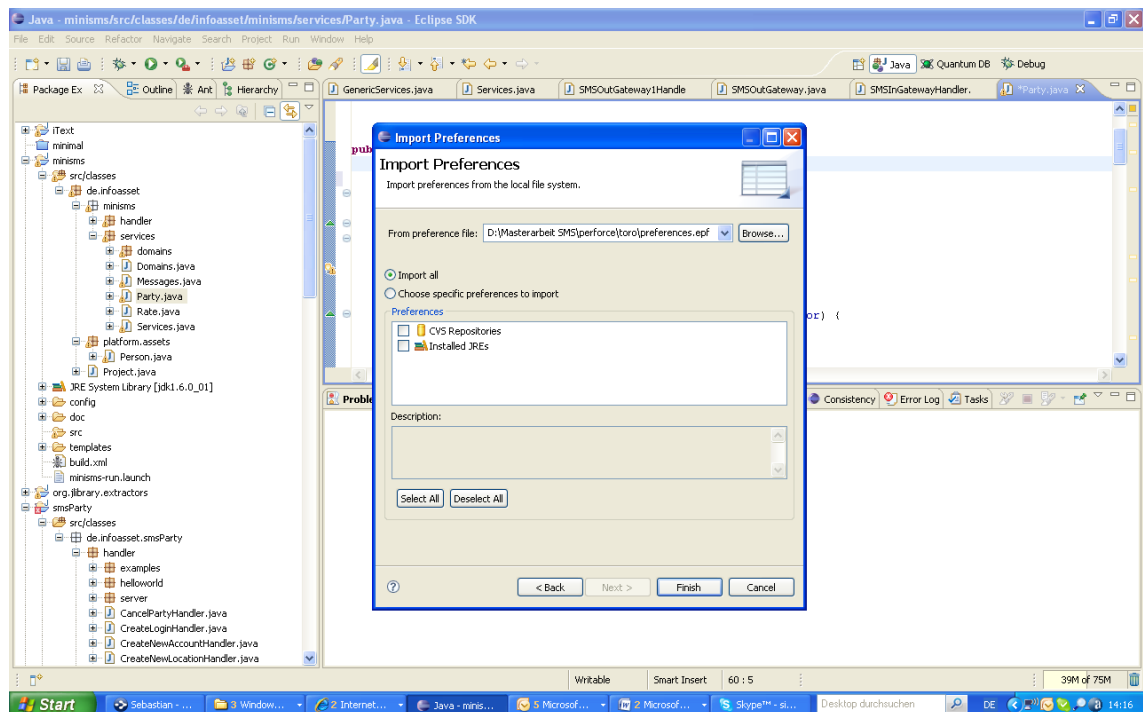


Figure 21: Importing Preferences into Eclipse

This preference file, however, will also change the color scheme of your source code. If you prefer to keep your own settings in this respect, then you can also restrict yourself to importing the templates, without the color scheme. This is done by choosing. Window->Preferences... and then Java->Editor->Templates:

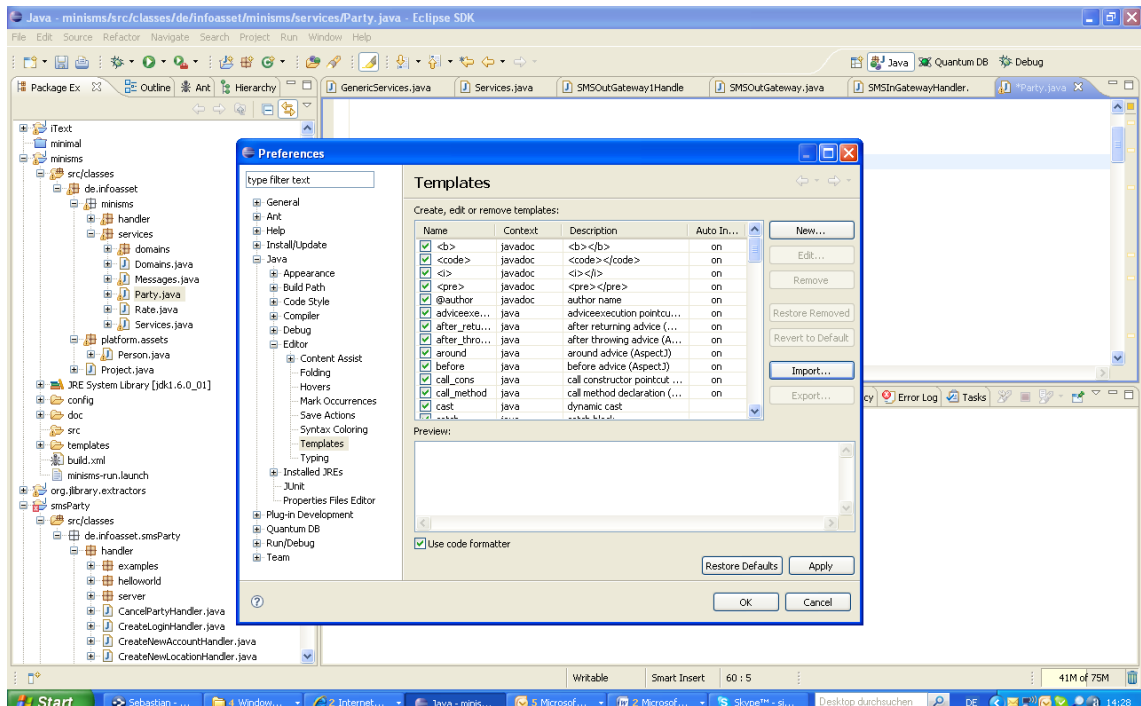


Figure 22: Importing Templates into Eclipse

Here, you choose the 'Import...' button and then select the `./toro/templates.xml` file. Either way, you are now ready to use Eclipse's Auto-Completion feature to enter Toro objects. For example, to define the **StringProperty**, as we have done in the above example, you now type: `property`, in the Java source file, and then press `[ctrl]+[space]`:

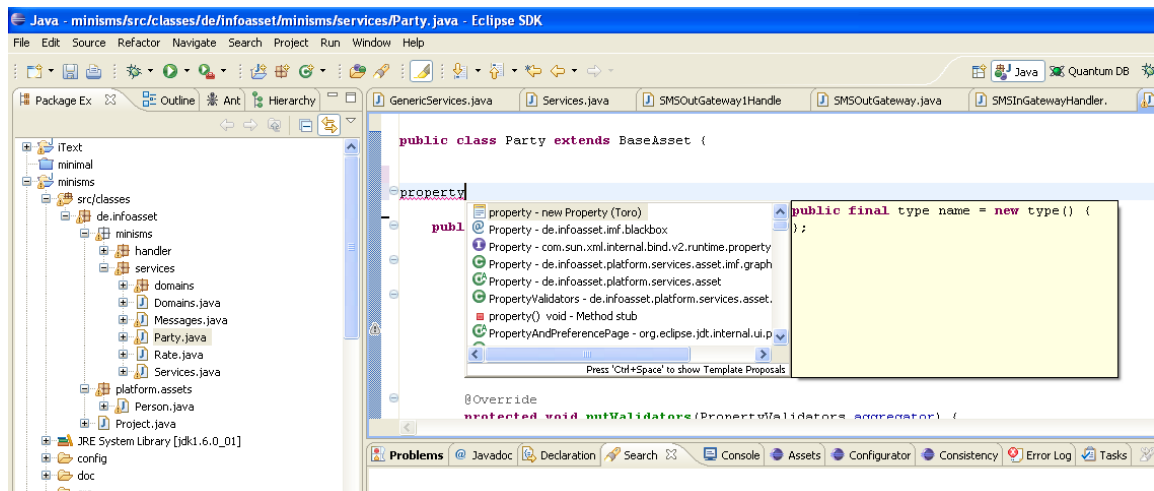


Figure 23: Choosing a template

The IDE-Template Engine will now propose a list of Substitutions – the first one on the list should be the Toro-Property Template. Choose this one:

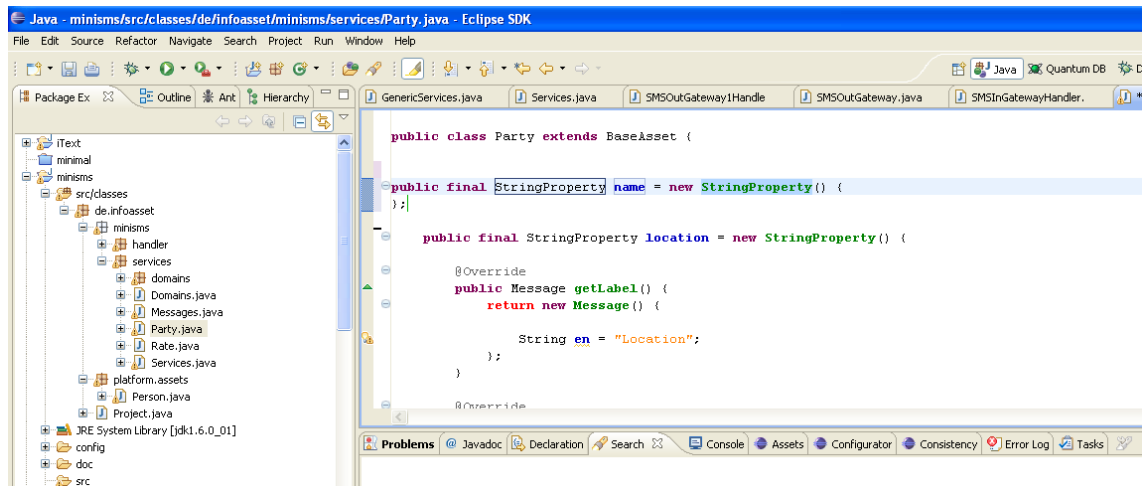


Figure 24: Filling in a template

With [tab], you can now enter the type of Property ('StringProperty') and the identifier. As you can see, Type and Constructor name, being the same, must now only be entered once.

The next thing we wish to tell the Asset about the 'name'-Property is its length restriction: 30 characters max. We do this by overwriting the **getMaxLength()** function of the **StringProperty** Class in the Anonymous Inner Class derived from it:

```

public class Party extends BaseAsset {
    public final StringProperty name = new StringProperty() {
        @Override
        public int getMaxLength() {
            return 30;
        }
    };
}

```

TIP: Overriding methods is, once again, best done using [ctrl] + [space], and typing in the first few letters of the method you wish to override:

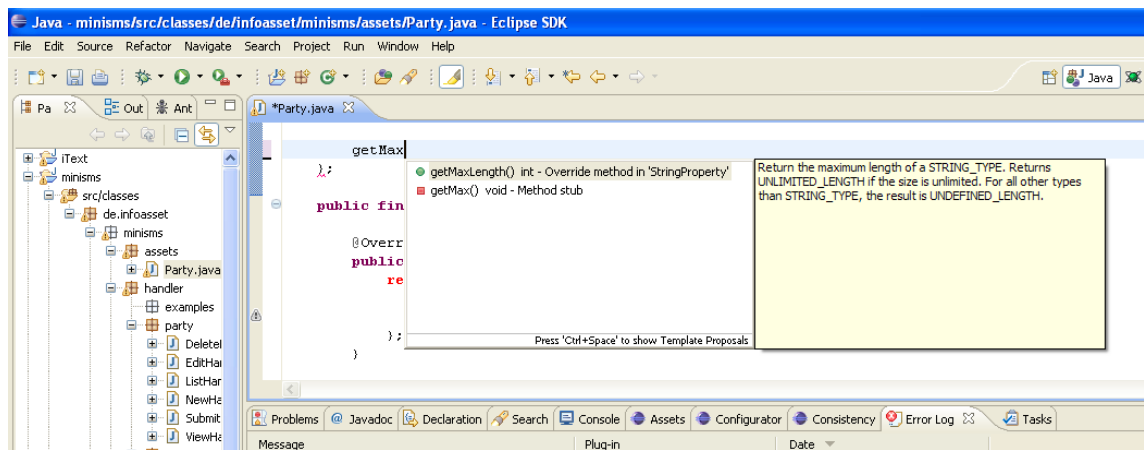


Figure 25: Choosing a Method with auto-completion.

Once the correct method has been chosen, type [return] to insert a stub in your source code. Then replace the stub with your code.

What else do we want to tell the 'name'-StringProperty? Two more things: What is the Property called, i.e. what label will we attach to this data when we display it on a web-page, and secondly, we want to force the user to enter this data. It is compulsory not optional, and null-values will not be accepted:

```
public class Party extends BaseAsset {

    public final StringProperty name = new StringProperty() {

        @Override
        public Message getLabel() {
            return new Message() {
                String en = "Name";
            };
        }

        @Override
        protected void putValidators(PropertyValidators aggregator) {
            aggregator.add(new NotNullValidator());
        }

        @Override
        public int getMaxLength() {
            return 30;
        }
    };
}
```

Labeling

Constraining

Labeling:

The mechanism for telling the 'name' **StringProperty** what label to attach is pretty straightforward: You overwrite a function called **getLabel()**, and in it return an object of type **Message**. The **Message** class has to be instantiated via the AIC mechanism once

again. This time, tuning the AIC is not done by overwriting an abstract method, but by defining a **String** called **'en'**, and assigning the value you wish to give to the Label (in our case: **'Name'**). **'en'** is the String attribute used to define English labels. You could also define a German label with the **'de'** String. In this way, we abstract the concrete labels, given in different languages, from the label itself. Should you wish to deploy your web-application in a different country, you will then only need to change a setting in your configuration, and all labels will henceforth appear in a different language. We will restrict ourselves to English examples.

Excursion :

“Anonymous Inner Classes”

The curled braces {} after **StringProperty()** will enclose all kinds of hints and customizations we will shortly wish to add to this *'name'* **Property**. Toro uses the mechanism of 'Anonymous Inner Class' (AIC) to accomplish customization (is it indexed?, how long can it be?, are numbers allowed?) of its Data Columns(=attributes) (In this case, a data column of type **StringProperty**). What we are actually doing is deriving a new class, a class derived from **StringProperty**, and instantiating it once. All of the customization of the derived class happens in the curly braces behind the Constructor of the (abstract class) **StringProperty()**, which we would normally not be allowed to call if it weren't for the curly braces, which indicate that not abstract **StringProperty**, but a concrete child should be instantiated – the concreteness of this child being defined inside of the braces.

Alternatively (as done in Hibernate), customization of the data-attributes of a persistent class could be accomplished through annotations – (lines beginning with a '@' as in: @this is an annotation), but the advantage of doing this through AICs is that the hints and customizations implemented through AICs are readily available to you as a programmer, not only to the persistence-engine, while annotations remain locked up and inaccessible to the programmer behind those '@' symbols. For more about AICs and Annotations, see [Ec06].

Constraining:

Telling the class not to take 'null' for an answer is a little more involved, because you hook into the **putValidators()** function. You can have several Validators for each property ('don't be null', 'consist only of letters', 'have at least five digits' etc.) One of Toro's features is a very flexible mechanism to fashion Validators of your choice, and stack them all on top of each other, forcing the Property you assign them to to comply with all of them. This 'stacking' is done inside of **putValidators()**, which stuffs all of these Validators into a list of type **PropertyValidators**. Which Validators we wish to apply is then done inside of **putValidators()** by **.add()**-ing new Validators to the list. As you can see, **putValidators()** is protected. This is difficult to remember, so in order to override functions in Toro, use Eclipse's auto-complete feature once you have the templates installed.

TIP: Finding out what kind of Classes Toro offers you as a programmer can be a challenge. Eclipse can help you. For example, you might be wondering what kind of Validators Toro offers. The thing to do is to look at the type hierarchy of a Validator you know exists, and then, by looking at its family tree, seeing what other appetizing Validators Toro has to offer. To do this, right click on the class reference whose relatives you wish to explore, and choose ‘Open Type Hierarchy’.

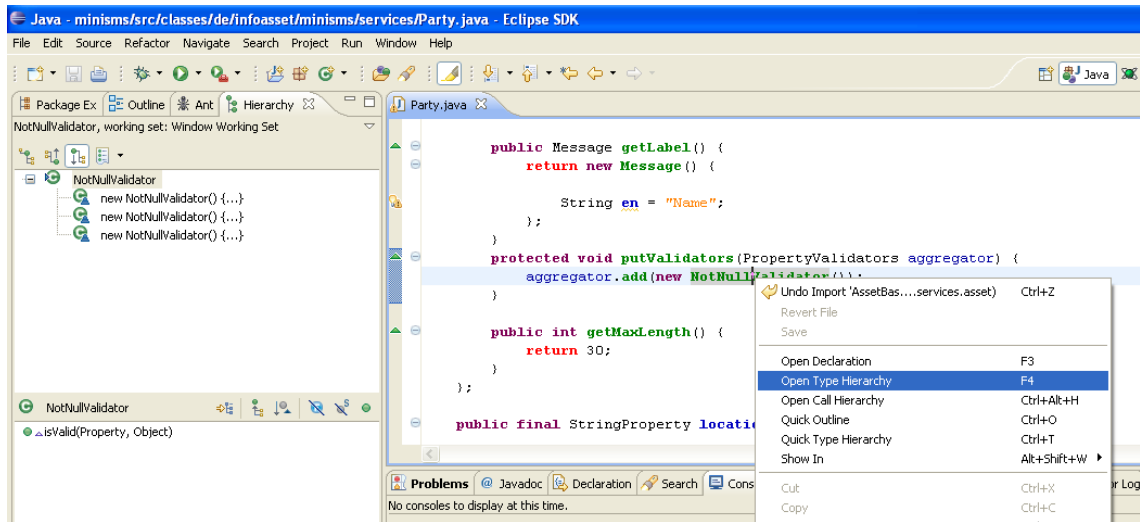


Figure 26: Displaying a classes’ subtypes

In the ‘Hierarchy’ tab, you will now see all the classes derived from **NotNullValidator()**. This is of limited interest: Three AIC’s have been derived, but what about the class’ forefathers? To display these, click on the ‘Supertype Hierarchy’ button in the ‘Hierarchy’ tab. (This button has a small arrow pointing up):

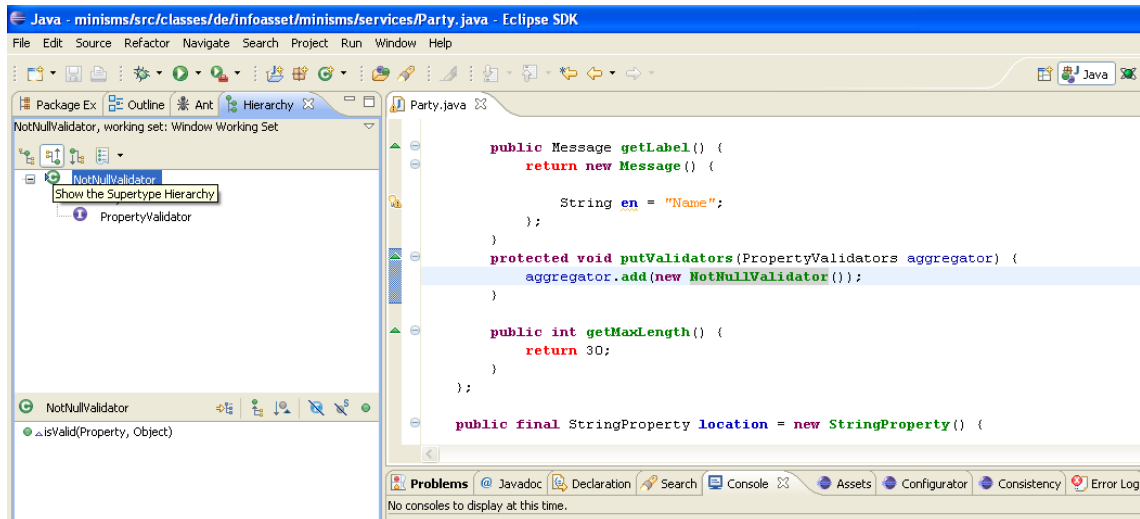


Figure 27: Displaying a classes’ pedigree

As you can now see, **NotNullValidator** implements an interface called **PropertyValidator**. This will be the common link between all Validators. DoubleClick on **PropertyValidator** in the type hierarchy to display the class file in the Edit-Window. Right-Clicking on **PropertyValidator** in the Edit-Window, once again choosing ‘Open Type Hierarchy’, and then displaying **PropertyValidator**’s subtypes in the

'Hierarchy-Tab' gives you an overview over all Validators, and scanning their .java files will give you a pretty good idea of their capabilities.

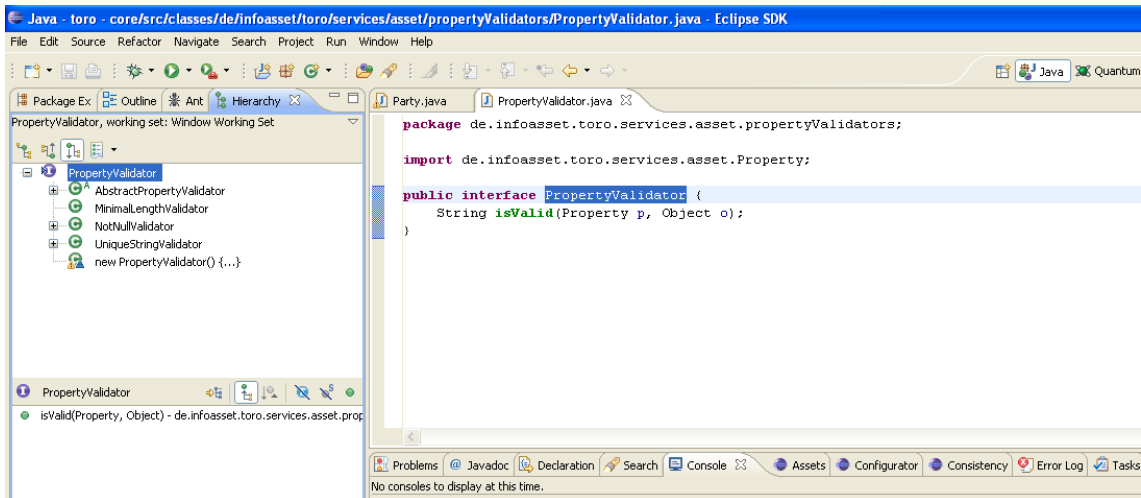


Figure 28: The hierarchy of PropertyValidators

Let us now add the 'location', 'size', 'price', 'begin' and 'purchaseDate' Properties of Party:

```
public class Party extends BaseAsset {

    public final StringProperty name = new StringProperty() {

        @Override
        public Message getLabel() {
            return new Message() {
                String en = "Name";
            };
        }

        @Override
        protected void putValidators(PropertyValidators aggregator) {
            aggregator.add(new NotNullValidator());
        }

        @Override
        public int getMaxLength() {
            return 30;
        }
    };

    public final StringProperty location = new StringProperty() {

        @Override
        public Message getLabel() {
            return new Message() {
                String en = "Location";
            };
        }

        protected void putValidators(PropertyValidators aggregator) {
            aggregator.add(new NotNullValidator());
        }
    };
}
```

```

    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final IntProperty size = new IntProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Size";
        };
    }

    @Override
    protected void putValidators(PropertyValidators aggregator) {
        aggregator.add(new NotNullValidator());
    }
};

public final IntProperty price = new IntProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Price";
        };
    }
}

public final TimestampProperty begin = new TimestampProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Begin";
        };
    }

    @Override
    protected void putValidators(PropertyValidators aggregator) {
        aggregator.add(new NotNullValidator());
    }
};

public final TimestampProperty purchaseDate = new TimestampProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Purchase Date";
        };
    }
};

public static final AssetSchema<Party> SCHEMA = new AssetSchema<Party>();
}

```

Integer Properties

Timestamp Properties

SCHEMA

This is basically more of the same:

Integer Properties:

Timestamp Properties:

As you can see, 'begin' and 'purchaseDate' are of class **TimestampProperty**, and 'size' and 'price' are of class **IntProperty**. We didn't put a **NotNullValidator** onto 'price' and 'purchaseDate', because these values will not be queried from the user through a web-form, but supplied by the system.

As an exercise, you can display the hierarchy of Property-Classes in the eclipse Hierarchy Tab, and see what different Properties are at your disposal.

SCHEMA:

The last line:

```
public static final AssetSchema<Party> SCHEMA = new AssetSchema<Party>();
```

is very important. Every Asset Class has an associated **AssetSchema** class, of which it has one single instance, which supplies services such as instantiation and gives information about the Class even if no instance is available. This last line of code generates this class and its instance, called 'SCHEMA'. Toro supplies you with a Generic **AssetSchema<>** class, which you then specialize to your Asset in the way shown above. If you are unfamiliar with the Classname followed by angle-brackets, you can read about Generics, a feature of Java SE5, in [Ec06]

3.5 Implementing Persistence – DomainValueProperties

We have not yet implemented the 'rateCategory' Property, which is of type 'Domain'. In our application, this attribute stores how expensive the party will be. We only want three price categories to exist. This means that we want 'rateCategory' to take only one value from a certain set of values. In this case, the Strings: 'cheap', 'medium' and 'expensive'.

Toro supplies the **DomainValueProperty** for this purpose:

```
public class Party extends BaseAsset {  
  
    public final StringProperty name = new StringProperty() {  
  
        @Override  
        public Message getLabel() {  
            return new Message() {  
                String en = "Name";  
            };  
        }  
    }  
}
```



```

@Override
protected void putValidators(PropertyValidators aggregator) {
    aggregator.add(new NotNullValidator());
}

@Override
public int getMaxLength() {
    return 30;
}
};

```

-
-

<snip>

-
-

```

public final DomainValueProperty rateCategory = new DomainValue-
Property() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Rate";
        };
    }

    @Override
    public Domain getDomain() {
        return Domains.RateDomain;
    }

    @Override
    public DomainValue getDefaultDomainValue() {
        return RateDomain.expensive;
    }
};

public static final AssetSchema<Party> SCHEMA = new AssetSche-
ma<Party>();
}

```

Specifying the Domain

Specifying a Default

Assigning a label to this new Property is the same as for all Properties, nothing new here.

Specifying the Domain:

DomainValueProperty has an abstract function you must overwrite when instantiating the AIC: **getDomain()**. In the return parameter of this function, you tell Toro what **Domain** you want this Property to hold. Of course, we must declare an appropriate Domain class before we can pass it on to Toro through **getDomain()**.

Specifying a Default:

You can also designate a default **DomainValue** by overwriting the **getDefaultDomainValue()** function in **DomainValueProperty**.

Domain classes must be declared in the package:

```
de.infoasset.minisms.services.domains
```

```
public class RateDomain extends Domain {  
  
    /* Define Domainvalues of RateDomain here */  
  
}
```

The individual **DomainValues** are once again instantiated through the AIC mechanism.

The **DomainValue** values have to be declared public static final. Each **DomainValue** requires a **nameMessage()** function which returns a **Message** instance, which we 'AIC' on the spot. You have seen how this works with the **getLabel()** function from **Property**, which also requires an instance of type **Message**.

```
public class RateDomain extends Domain {  
  
    public static final DomainValue expensive = new DomainValue() {  
  
        @Override  
        protected Message nameMessage() {  
            return new Message() {  
                String en = "expensive";  
            };  
        }  
    };  
}
```

It is important to understand that the String which will be written into the database when a **DomainValueProperty** is written will *not* be the String declared inside of **nameMessage**. The value written into the database is gleaned from the reference name assigned to the **DomainValue**. The country coded strings are only shown when representing this value on a web page.

The other two rates are added to **RateDomain** in the same way:

```
public class RateDomain extends Domain {  
  
    public static final DomainValue expensive = new DomainValue() {  
        @Override  
        protected Message nameMessage() {  
            return new Message() {  
                String en = "expensive";  
            };  
        }  
    };  
}
```

```

        };
    }
};

public static final DomainValue medium = new DomainValue() {
    @Override
    protected Message nameMessage() {
        return new Message() {
            String en = "medium";
        };
    }
};

public static final DomainValue cheap = new DomainValue() {
    @Override
    protected Message nameMessage() {
        return new Message() {
            String en = "cheap";
        };
    }
};
}

```

We are almost done. We still have to register the newly created Domain with Toro. In the `de.infoasset.minisms.services` package you will find a Class called **Domains**. You have to add a public final static **Domain** reference to this class, and set it to an instance of your new Domain:

```

public class Domains extends GenericDomains {
    public static final Domain Title = new Title();
    public static final Domain RateDomain = new RateDomain();
}

```

(‘*Title*’ is a Domain left over from ‘minimal123’. Right after that, we have now registered our new Domain ‘*RateDomain*’.)

As you can see in the listing of the **Party** class, the `getDomain()` function of a **DomainValueProperty** returns the instance of **RateDomain** referenced in **Domains**.

3.6 Implementing Persistence – The ‘Person’ Class

Every web-application is bound to have something like a User-Account. In fact, Toro is so convinced of this that it actually supplies you with a prefabricated Asset Class called **Person**. You can overwrite this class to suit your needs – but you cannot change its

name or eliminate it. Let us look at the data structure we wish to implement for our **Person** class:

Person
+eMail: String (maximum size = 100) +firstName: String (maximum size = 30) +lastName: String (maximum size = 30) +street: String (maximum size = 30) +zipCode: String (maximum size = 10) +city: String (maximum size = 30) +country:String (maximum size = 30) +password: Password (maximum size = 30) +membershipDate:Date

Figure 29: Structure of the Person Asset.

Because we wish to replace the **Person** class preinstalled by Toro we, must place this new Asset in a very specific package in our project. So please create a new Class called **Person** derived from **BaseAsset** in the package `de.infoasset.platform.assets`. In which order you place Properties is, by the way, completely up to you. User Editable Properties should be listed in the sequence you want the User to edit them:

```

public class Person extends BaseAsset {

    public final BooleanProperty isAdministrator = new BooleanProperty() {

        @Override
        public boolean getDefaultValue() {
            return false;
        }

        @Override
        public boolean hasDefaultValue() {
            return true;
        }

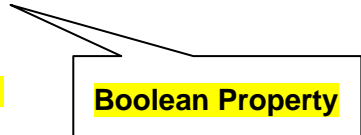
        @Override
        public Message getLabel() {
            return new Message() {

                String en = "Is this Person an Administrator?";
            };
        }
    };

    public final StringProperty eMail = new StringProperty() {

        @Override
        public Message getLabel() {

```



```

        return new Message() {
            String en = "E-Mail Address";
        };
    }

    @Override
    public int getMaxLength() {
        return 100;
    }

    @Override
    protected void putValidators(PropertyValidators aggregator) {
        aggregator.add(new NotNullValidator() {});
        aggregator.add(new RegExpValidator() {
            @Override
            protected Message getErrorMessage() {
                return Messages.validator invalid e mail;
            }

            @Override
            public String getRegExp() {
                return "^[a-z0-9-]+(\\.[a-z0-9-]+)*@[a-z0-9-
]+(\\.[a-z0-9-]+)*$";
            }
        });
    }
}

public final StringProperty firstName = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {
            String en = "First Name";
        };
    }

    @Override
    public boolean isFulltextIndexed() {
        return true;
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final StringProperty lastName = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Last Name";
        };
    }
};

```

Regular Expression Validator

Switch-Funtion

```

    };
}

@Override
protected void putValidators(PropertyValidators aggregator) {
    aggregator.add(new NotNullValidator());
    aggregator.add(new MinimalLengthValidator(3));
}

@Override
public boolean isFulltextIndexed() {
    return true;
}

@Override
public int getMaxLength() {
    return 30;
}
};

public final StringProperty street = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "Street";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final StringProperty zipCode = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "ZipCode";
        };
    }

    @Override
    public int getMaxLength() {
        return 10;
    }
};

public final StringProperty city = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

```

Minimal Length Validator

```

        String en = "City";
    };
}

@Override
public int getMaxLength() {
    return 30;
}

};

public final StringProperty country = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "Country";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }

};

public final StringProperty password = new PasswordProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "Password";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }

};

public final TimestampProperty membershipDate = new TimestampProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "Date of Membership";
        };
    }

};

```




Password Property

```

    public static final AssetSchema<Person> SCHEMA = new AssetSche-
ma<Person>() {
    };

    public boolean isAdministrator() {
        return isAdministrator.get();
    }
}

```



Most of this is nothing new. So let us restrict our attention to the highlighted areas, which have something original:

Boolean Property and isAdministrator():

The **BooleanProperty isAdministrator** and the Function **isAdministrator()** are a legacy from the original **Person** class. In order not to confuse Toro, they should be retained, even if you do not wish to distinguish between Administrators and normal Users. If you have not yet explored the different kinds of available Properties, **BooleanProperty** will be new to you. The name is pretty self-explanatory. As you can see here, access to the value of a Property is done by using the **.get()** function on the Property attribute, as in:

```
isAdministrator.get();
```

Another interesting feature seen inside the **BooleanProperty** class is the use of the functions **hasDefaultValue()** and **getDefaultValue()**. All Properties can be preconfigured using these functions. **hasDefaultValue()** should always return 'true'³, and **getDefaultValue()** returns the default value itself, in this case a Boolean, but a String in case of **StringProperty**, a Date in case of **DateProperty** etc.

Regular Expression Validator:

RegExpValidator is the most powerful and versatile Validator available in Toro. In the AIC definition, you overwrite the Function **getRegExp()**, in which you have to return the Regular Expression the Property value should comply with. Regular Expressions are a programming language in their own right, and describe what symbols in which sequence a term can have. A good reference is [Go08]. In our case the Regular expression makes sure there is a '@' Symbol in the email address, and no characters are used which are illegal for email addresses.

RegExpValidator has an abstract function **getErrorMessage()** you must overwrite. Here, you supply a **Message** which will be displayed in case your user fails to comply with the Regular Expression when entering data. You can of course AIC your own Message on the spot, but, actually, we have a whole collection of standard Error Messages at our disposal: In the package `de.infoasset.miniparty.services` you will

³ You could return 'false'. However, in this case I would recommend simply not defining `hasDefaultValue()`.

find a class called **Messages**, containing diverse Messages for all purposes, and `'Messages.validator_invalid_email'` is very appropriate in our case.

Minimal Length Validator:

Passwords and Last Names should not be too short. We can enforce this with a **MinimalLengthValidator**. In this case, the minimal character count is passed to the Constructor of the **MinimalLengthValidator** as a parameter.

Password Property:

This Property is very similar to **StringProperty**. However, input fields associated with this type will display characters entered as 'stars' (*) to hide them from view, and query them twice, to check for typos. These are two features we want here, because 'password' is just that: A password, and it should not be displayed or misspelled.

3.7 Implementing Persistence – Relationships between Assets

Right now, we have two isolated Tables in the Database, without any Relationship. Our next step will be to connect them. In Figure 9) you can see that one person can organize several parties, but that each party has one, and only one, person associated with it. Modeling this relationship requires the introduction of a new kind of Property in our two Assets, called **Role**. Roles come in two flavors: **ManyRole<>** and **OneRole<>**. In **Person**, we will have a **ManyRole<Party>** called 'parties' (the name, of course, is our choice, but 'parties' is a good one), and in **Party** we will have a **OneRole<Person>** called 'organizer'. Both **Role** Properties are Generics [Ec06], and must be specialized to the Asset Class they refer to.

Each **Role** class requires you to overwrite a Function called **otherRole()**. In it, you return an instance of the Role on the other side of the association: In the 'parties' **ManyRole** of the **Person** class, you will return the 'organizer' **OneRole** of the **Party** class, and vice versa.

Let us first add the **ManyRole<Party>** to **Person**:

```
public class Person extends BaseAsset {  
  
    public boolean isAdministrator() {  
        return isAdministrator.get();  
    }  
  
    public final BooleanProperty isAdministrator = new BooleanProperty()  
}
```

```

public boolean hasDefaultValue() {
    return true;
}

public boolean getDefaultValue() {
    return false;
}

public Message getLabel() {
    return new Message() {
        String en = "Is this Person an Administrator?";
    };
}
};

```

•
•
<snip>
•
•

```

public final TimestampProperty membershipDate = new TimestampProperty() {

```

```

    public Message getLabel() {
        return new Message() {
            String en = "Date of Membership";
        };
    }
};

```

ManyRole

```

/* Person 1 <-> * Party */
final public ManyRole<Party> parties = new ManyRole<Party>() {
    public Role otherRole() {
        return Party.SCHEMA.prototype().organizer;
    }
};

```

```

public static final AssetSchema<Person> SCHEMA = new AssetSchema<Person>() {
};

```

}

ManyRole:

The **ManyRole<>** is AIC'ed, and specialized to **Party**, the other end of the relationship.

As you can see in `otherRole()`, returning the 'organizer' Role is not completely trivial. After all, we don't have an instance of `Party` at our disposal, from which we could access the 'organizer' Role. It is here that the public static '`SCHEMA`' object comes in handy. (Right beneath the 'parties' Role you can see the `SCHEMA` of `Person` being instantiated. The same thing has been done with `Party`.) The `Party.SCHEMA.prototype()` function then supplies a (generic) 'missing instance' of `Party`, through which we can then access the 'person' Role. Don't worry about Eclipse momentarily complaining about there not being such a reference. We will now add it to the `Party` class, in the same way we have just done with `Person`:

```
public class Party extends BaseAsset {

    public final StringProperty name = new StringProperty() {

        public Message getLabel() {
            return new Message() {

                String en = "Name";
            };
        }

        protected void putValidators(PropertyValidators aggregator) {
            aggregator.add(new NotNullValidator());
        }

        public int getMaxLength() {
            return 30;
        }
    };
};
```

•

•

<snip>

•

•

```
    public final DomainValueProperty rateCategory = new DomainValue-
Property() {

    public Message getLabel() {
        return new Message() {

            String en = "Rate";
        };
    }

    public Domain getDomain() {
        return Domains.RateDomain;
    }

    public DomainValue getDefaultDomainValue() {
        return RateDomain.expensive;
    }
};
```

```

/* Party * <-> 1 Person */
final public OneRole<Person> organizer = new OneRole<Person>() {
    public Role otherRole() {
        return Person.SCHEMA.prototype().parties;
    }
};

```

```

    public static final AssetSchema<Party> SCHEMA = new AssetSche-
ma<Party>();
}

```

Adding the highlighted code to your **Party** class should fix the Eclipse-Complaint.

Connecting Assets can be a bit confusing at first. Remember: Start with the class you want to have connected. Then ask: What do I want to connect to? The answer to this question is the Type of Role you need to embed as a Role. Which one: A **OneRole** or a **ManyRole**? If there is more than one Asset on the far side of the connection, it is a **ManyRole**, otherwise, it will be a **OneRole**. With what Role do I have to overwrite the **otherRole()** function? With myself, the Asset I am connecting from, and in which I am embedding the Role. So I return the Role pointing to me from the other side of the connection.

3.8 Implementing Persistence – Registering your Assets

We are almost done with the persistence level of our application. There remains one last act of bookkeeping: All of the assets must be registered in the ‘**Services**’ class located in the package `de.infoasset.minsms.services`. As you can see there, the ‘**Person**’ class has already been registered, we must now do the same thing for the ‘**Party**’ class:

```

public class Services extends GenericServices {
    private static Services instance;

    public static Services INSTANCE() {
        return instance;
    }

    protected void initData() {
        Person admin = Person.SCHEMA.findSingleAsset(new QueryE-
quals(Person.SCHEMA.prototype().isAdministrator, true));
        if (admin == null) {
            admin = Person.SCHEMA.createAsset();
            admin.eMail.set(adminMail);
            admin.firstName.set("Max");
            admin.lastName.set("Mustermann");
            admin.isAdministrator.set(true);
            admin.acceptsConditions.set(true);
            admin.password.set("ottto");
            admin.title.set(Title._mr);

```

Delete this!

```

        GenericAssetListener.commit();
    }

    protected void initSchemas () {
        initSchema(Person.class);
        initSchema(Party.class);
    }
}

```

Add this!

Don't worry about the overhead for the moment: **initData()** is a function called in case you have configured your application to start with a new slate of data each time. It is a good place to fill our Database with test-data. However, the test-data in the present form is a relic of the 'minimal123' project we started out with, so we delete it. We will be filling the database with our own test-data in the next chapter. In the last line, we have added `initSchema(Party.class)`; This is all it takes to register the new Asset. The Persistence Level of our web-application is now complete, and ready for use.

3.9 Implementing Persistence – Persisting Data

We will now proceed by replacing the recently deleted test-data left over from the 'minimal123' project with some of our own test data. Let us feed the database with an instance of **Person**, then an instance of **Party**, and then connect the two. So open the **Services** class located in the package `de.infoasset.minisms.services` once again, and add the highlighted code:

```

public class Services extends GenericServices {
    private static Services instance;

    public static Services INSTANCE() {
        return instance;
    }

    protected void initData () {
        Person tester = Person.SCHEMA.createAsset();

        tester.eMail.set("mymail@myserver.com");
        tester.firstName.set("Sebastian");
        tester.lastName.set("Henckel");
        tester.street.set("Kaiserdamm 28");
        tester.zipCode.set("14057");
        tester.city.set("Berlin");
        tester.country.set("Germany");
        tester.password.set("prettyplease");
        tester.membershipDate.set(new Date(108, 0, 1));
        tester.isAdministrator.set(false);

        Party fiesta = Party.SCHEMA.createAsset();

        fiesta.name.set("Test Party");
    }
}

```

Creating

Setting Values

```

fiesta.location.set("Irish pub");
fiesta.size.set(100);
fiesta.begin.set(new Date(108, 0, 1));
fiesta.rateCategory.set(RateDomain.cheap);
fiesta.purchaseDate.set(new Date());

tester.parties.create(fiesta);

GenericAssetListener.commit();
}
protected void initSchemas() {
    initSchema(Person.class);
    initSchema(Party.class);
}
}

```

The diagram shows two callout boxes. The first box, labeled 'Connecting', points to the line `tester.parties.create(fiesta);`. The second box, labeled 'Persisting', points to the line `GenericAssetListener.commit();`.

Creating:

SCHEMA.createAsset() is used to 'construct' a new instance of an Asset Class. The reason why this is not done through a constructor are technical, and internal to Toro: Through introspection, Toro has to build code inside of the Engine that will assure Database-Persistence for all of the Property Instances of the new Asset. All you have to remember is to instantiate new instances of Assets through **createAsset()** and not through a Constructor. Using a Constructor will create unpersistent Assets, and could cause serious trouble.

Setting Values:

Writing values to Properties of Assets is done through the `[Asset].[Property].set([value])` function. **set()** will take a type appropriate to the Property you are trying to set. For `'password'` this will be a String, for `'membershipDate'` a Date, for `'isAdministrator'` a Boolean Value.

Connecting:

Having instantiated a **Person** (called `'tester'`) and a **Party** (called `'fiesta'`), a connection between these two must now be created. This is done through the line of code:

```
tester.parties.create(fiesta);
```

Alternatively, we could also have written:

```
fiesta.organizer.create(tester);
```

You only need to do one of these. Toro will worry about connecting the other end of the relationship for you.

Persisting:

Data written into Properties with **.set()** is not immediately saved in the Database. Instead, you are allowed to have inconsistent states of your Assets. This is a good thing,

because otherwise, Toro would complain immediately after instantiating a new Asset through **createAsset()**, as no values have yet been written to the Properties, and many Properties will not comply with their Validators. Toro persists Assets automatically when leaving a web-Page, so make sure your Assets are consistent before this happens (An exception to this rule is '**TransientAsset**', a type of Asset you will encounter later on in the tutorial). Right now, we are not yet in the realm of web-Pages, so we need to externally trigger a write, without leaving a web-Page: **GenericAssetListener.commit()** does just that.

3.10 Implementing Logic – Pageful Handlers

We now proceed to implement the Logic and Presentation Strata of our application.

In Toro, these two are closely connected: Every HTML-web Page has a Java-Class associated with it, and when you access the HTML-Page through your browser, the first thing Toro does is to execute the methods of this Java-Class, which is called a Handler. It is here that the Logic of your application will take place. Afterwards, the webpage is displayed, and only then do we reach the Presentation Layer. There is a simple naming law connecting web-Pages with Handlers: If the webpage is called:

examplePage.htm

The handler must be called:

ExamplePageHandler.java

Handlers and web-Pages must be placed in a specific package, respectively in a specific folder, of your project. The naming and location rules are summarized in the following table:

	Name	Location
HTML File ("Page")	examplePage.htm	\minisms\templates\standard\
Java File ("Handler")	ExamplePageHandler.java	de.infoasset.minisms.handler

Table 4: Naming Law for Pages and Handlers

If you want to, you can place corresponding Handlers and web-Pages in equally named sub-packages and sub-directories of the locations shown in this table. This is a good convention, because in large projects, Handlers and Pages dealing with completely different aspects of your web-application will otherwise clutter up your project. Generally, you will find that dedicating one package for Handlers and one corresponding folder for Pages *per Asset* is a good practice. In this tutorial, we will have two folder/package sets:

```
\minisms\templates\standard\party\  
de.infoasset.minisms.handler.party
```

for the Asset 'Party', and

```
\minisms\templates\standard\person\  
de.infoasset.minisms.handler.person
```

for the Asset 'Person'.

Always make sure that Handler-package and Page-folder *correspond*.

TIP: The naming rule connecting Pages and Handlers is a *law*, but the practice of placing Handlers and pages belonging to an Asset in dedicated packages and folders is a *convention*, which you can ignore. I will try to make it clear in this tutorial when we are talking about laws, and when we are merely dealing with conventions.

Before actually doing anything useful, let us build a small web-page called **testing.htm**, with an associated Handler-Class called **TestingHandler.java**, and see how this works:

Create the following **testing.htm** file in the `\minisms\templates\standard\` directory of the project:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <meta http-equiv="Content-Type" content="text/html; char-  
set=UTF-8"/>  
    <script type="text/javascript" src="/_/jquery.js"></script>  
    <script type="text/javascript" src="/_/platform.js"></script>  
    <link rel="stylesheet" type="text/css" href="/_/Toro-  
platform.css"/>  
    <title>Toro Tutorial</title>  
  </head>  
  <body>  
    <div>  
      <h1>TESTING</h1>  
    </div>  
  </body>  
</html>
```

This is a basic HTML Page [Se08].

The highlighted HTML code binds two Java-Script libraries and a CSS stylesheet into your code which Toro depends upon (not in this simple 'Test-Page', but later on).

TIP: The Toro project 'platform-toro' which was imported into Eclipse during the installation is a 'fall-back project' which the framework application looks to if you have not supplied an Asset/Script/StyleSheet etc. of your own. In case of the 'Person' **BaseAsset**, we chose to define our own version, and overrode the Asset supplied by 'toro-platform'. In case of jquery.js, platform.js and Toro-

platform.css, we wish to fall back on the resources supplied by 'platform-toro', so no further action is needed. Toro will look for the files in: \minisms\templates\standard__\ first. Not finding them there, it will then look in \platform\templates\standard__\.

Next, create the corresponding Handler file **TestingHandler.java** class in the `de.infoasset.miniparty.handler` package of the project:

```
public class TestingHandler extends Handler {  
  
    final SimplePage TESTING_PAGE = new SimplePage() {  
    };  
  
    public Station doBusinessLogic() throws Exception {  
        System.out.println("TestingHandler dBL()");  
        return TESTING_PAGE ;  
    }  
}
```

Let us dissect this code line by line:

First, all Handlers will inherit from the class **Handler**. We already talked about the naming law: Capitalize the HTML File-Name and append '-Handler'.

Next comes an AIC of type **SimplePage**. Every pageful Handler will have one of these. In order to make Toro happy, please make the reference 'final'. What is this **SimplePage** object (which we could have called anything, but we chose to call '*TESTING_PAGE*')?

It represents the HTML page the browser is supposed to show. By overwriting functions in it, we can tell Toro about run-time substitutions we want to have performed inside the HTML page. In this particular page, we don't want anything to be done: The **testing.htm** is supposed to be shown just the way it is. And because of the naming law, we don't even have to tell Toro how it is called. Toro can figure this out from the Class Name.

Next, we will discuss the **doBusinessLogic()** function which every Handler has. It is here that you can do whatever you feel like – this is where your application is individualized. In our case, we just print out some text on the Console. In the end, we return the **SimplePage** we want to have displayed.

Now let us run this application. Run 'minisms' in Eclipse. The Eclipse Console will prompt you whether you really want to reset the Database, which we affirm by typing in 'Y':

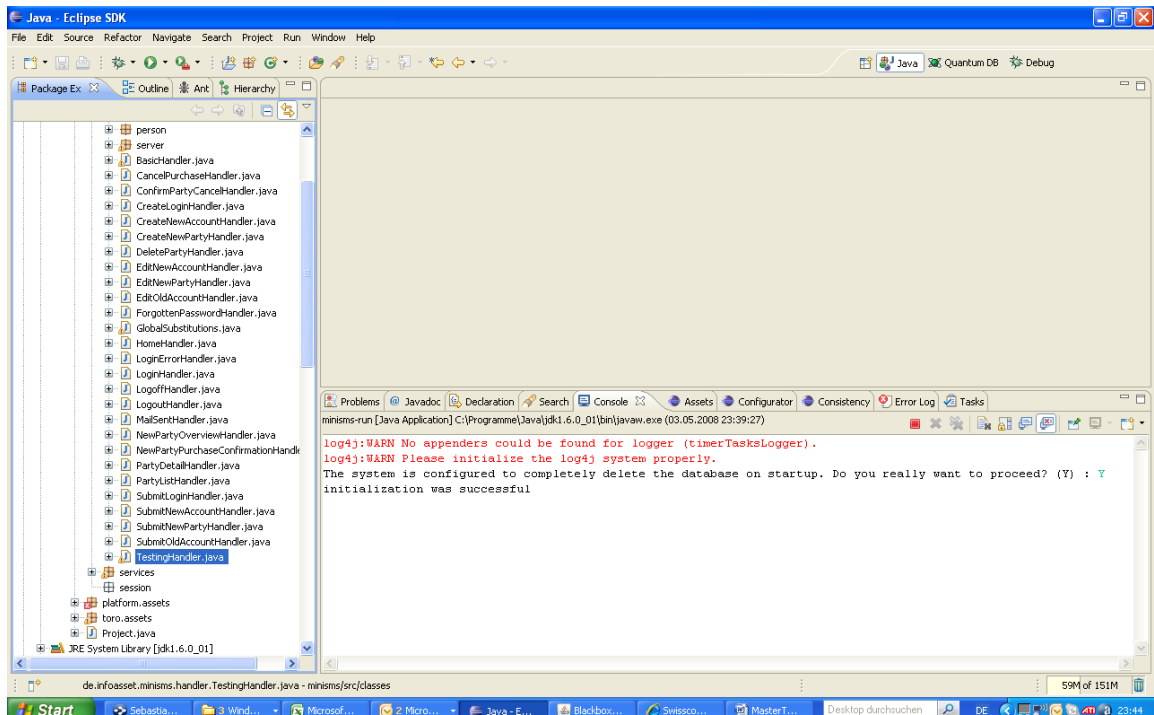


Figure 30: Launching the Toro application

Toro is now running, and ready to display the 'testing.htm' page. Enter:

<http://localhost:8083/testing>

Into your browser URL-field.

This is what you should see:

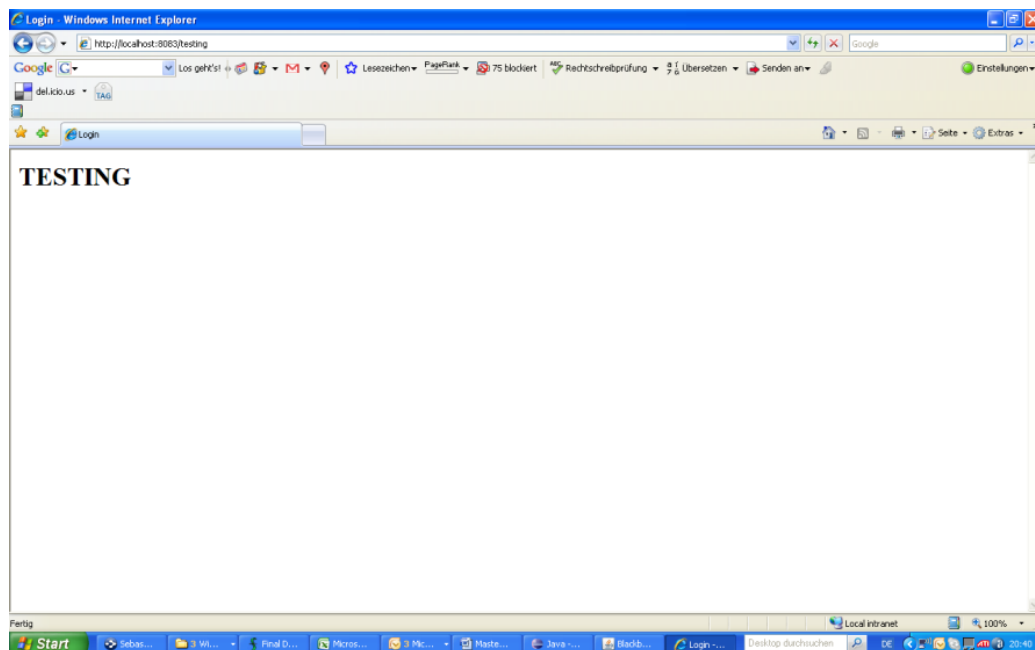


Figure 31: A simple web-page with Toro

Checking the ‘Console’ Tab in Eclipse, you will also see that **doBusinessLogic()** has faithfully output: “TestingHandler dBL()”.

3.11 Implementing Logic – Pageless Handlers

A web-application does not only consist of HTML pages and their corresponding Handlers. Sometimes, you will wish to perform application-logic without displaying a web-page. Of course, in the end, a web-page will be displayed, but which one might depend on user-input, or the state of your application. For this purpose, Toro has ‘Pageless Handlers’: These are also Handlers, but they do not have a **SimplePage** associated with them. Instead, they have one or several instances of the AIC **Line**. A **Line**, as the name suggests, is a connection to another Handler, which will take over from the pageless Handler. This Handler can be another pageless Handler, or a pageful Handler. **Lines** and **SimplePages** are both subclasses of the more general class **Station**, the type **doBusinessLogic()** is expected to return. In case of a pageless Handler, **doBusinessLogic()** returns an instance of **Line**, in the pageful Handlers an instance of **SimplePage**. An example for this is the login process for our web-application. Once the login-form has been read, we check to see whether the login was successful – in which case we proceed to the ListParty page, or whether not: In this case, we return to the Login page:

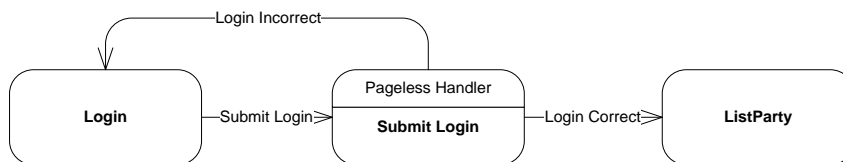


Figure 32 Processing a Login Form with a pageless Handler

As you can see in Figure 32) the “weigh-station” between the pageful ‘Login’ and ‘List-Party’ pages is the pageless ‘SubmitLogin’ page. Let us now create these three Handlers and their corresponding pages.

First, let us create **login.htm**:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; char-
set=UTF-8"/>
    <script type="text/javascript" src="/__/jquery.js"></script>
    <script type="text/javascript" src="/__/platform.js"></script>
    <link rel="stylesheet" type="text/css" href="/__/Toro-
platform.css"/>
    <title>Toro Tutorial</title>
  
```

```
</head>
<body>
```

You must login....

```
<p>
  <form action="/submitLogin">
    Login: <input type="text" name="login"/>Password: <input
type="password" name="password"/>
    <input type="submit" value="Submit"/>
  </form>
</p>
</body>
```

This is a simple HTML page with a form containing two fields: A Login, and a Password. Save this file outside of an Asset Folder. It does not belong to any specific Asset:

```
\minisms\templates\standard\login.htm
```

Next, let us create a placeholder for the Party List. Later on, we will show how to retrieve the parties associated with a specific person from the Database, and list them. Right now, however, we want to learn how to use pageless Handlers, and just need a dummy.

```
\minisms\templates\standard\party\list.htm:
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; char-
set=UTF-8"/>
    <script type="text/javascript" src="/__/jquery.js"></script>
    <script type="text/javascript" src="/__/platform.js"></script>
    <link rel="stylesheet" type="text/css" href="/__/Toro-
platform.css"/>
    <title>Toro Tutorial</title>
  </head>
  <body>
```

May parties will be listed here....

```
</body>
```

This file is clearly associated with the 'Party' Asset, and should be stored appropriately in the party folder.

To finish this up, let us also create the corresponding dummy Handler. It must be located in the package `de.infoasset.minisms.handler.party`, and called **ListHandler**:

```
public class ListHandler extends Handler {
```

```

    final SimplePage LISTPAGE = new SimplePage() {
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        return LISTPAGE;
    }
}

```

We now write the Handler associated with the Login Page. This is a pageful Handler, containing an instance of **SimplePage**. It does absolutely nothing. However, do not believe that you can therefore omit the Handler. Toro still expects every page to have a Handler, and will not work without one.

```

public class LoginHandler extends Handler {

    final SimplePage LOGINPAGE = new SimplePage() {
    };

    public Station doBusinessLogic() throws Exception {
        return LOGINPAGE;
    }
}

```

Save this Handler in the Handler location corresponding to:

```
\minisms\templates\standard\login.htm
```

which is the package: `de.infoasset.minisms.handler`

Unsurprisingly, the Handler associated with the List Page is very similar, and just as boring:

```

public class ListHandler extends Handler {

    final SimplePage LIST_PAGE = new SimplePage() {
    };

    public Station doBusinessLogic() throws Exception {
        return LIST_PAGE;
    }
}

```

However, you have to place it in a different package, corresponding to the 'Party' Asset:

```
de.infoasset.minisms.handler.party
```

Let us now come to the topic of this chapter, the pageless Handler, which is called when we push the Submit Button of the login.htm form. As you can see in the HTML code, pushing the submit button forwards to a (non-existent) HTML page called **submitLogin**. Following the naming law, Toro will look for a Handler called **SubmitLoginHandler**. This is the pageless Handler. Just as the LoginHandler, this Handler is also

not associated with a specific Asset, and should therefore be placed in the `de.infoasset.minisms.handler` package.

```
public class SubmitLoginHandler extends Handler {  
  
    private String login;  
  
    private String password;  
  
    @Override  
    public void getParameters(ParameterReader parameters) {  
        login = parameters.getString("login");  
        password = parameters.getString("password");  
    }  
  
    final Line SUCCESS = new Line() {  
        @Override  
        public void next(Forwarder f) {  
            f.go(ListHandler.class);  
        }  
    };  
  
    final Line FAILURE = new Line() {  
        @Override  
        public void next(Forwarder f) {  
            f.go(LoginHandler.class);  
        }  
    };  
  
    private Person candidate;  
  
    private boolean noLogin;  
  
    private boolean notRegistered;  
  
    private boolean pwdError;  
  
    @Override  
    public Station doBusinessLogic() throws Exception {  
        if (login == null || login.length() == 0) {  
            noLogin = true;  
            return FAILURE;  
        }  
        candidate = Person.SCHEMA.findSingleAsset(new QueryE-  
quals(Person.SCHEMA.prototype().email, login));  
        if (candidate == null) {  
            notRegistered = true;  
            return FAILURE;  
        }  
        if (!candidate.password.get().equals(password)) {  
            pwdError = true;  
        }  
    }  
}
```

Get Parameters

Success Line

Failure Line

Retrieve Customer from DB


Failure

Failure

```

        return FAILURE;
    }
    GenericSessionLocal.getSession().setUser(candidate);
    return SUCCESS;
}
}

```



In respect to the declaration line, a pageless Handler does not look any different from a pageful Handler: Both are derived from **Handler**.

Get Parameters:

Every Handler can – but is not obliged to – have a function called **getParameters()**. If this function is defined by you, the programmer, the framework will call it for you *before anything else happens*. The Framework feeds it an argument of Type **ParameterReader**, which will contain all the parameters passed to the Handler via HTTP from the previous page. We always come to this Handler via the ‘Login’ Page, and the form submitted in this page passed two parameters, called ‘login’ and ‘password’. We assign them to the two private Strings ‘login’ and ‘password’, fetching them from the **ParameterReader** with the method **.getString()**, which, as an argument, takes the name of the parameter passed via the GET method.

Retrieve Customer from DB:

We will have much more to say about Database Access in later chapters of this tutorial, but the Logic of this Handler – deciding whether to grant access to the system or not – cannot be implemented without checking with the Database whether the user has entered the correct login and password. To do this, we query the Database by creating a **Query** Object. Several different Flavors of Queries exist – you can check on them in Eclipse’s Hierarchy window. Right now, we want to see if there is a customer whose email address – which is also uses as login – equals the login given in the login page:

```

Person.SCHEMA.findSingleAsset(new QueryEquals (Person.SCHEMA.prototype().eMail, login))

```

The **QueryEquals** object is created with a Constructor taking two parameters: The first one is a **Property** of a certain **BaseAsset** we want to compare to a value. The second one is the value we want to compare it with. In order to supply the first argument, we need an instance of **Person**, something we don’t have at this time. We can get a ‘dummy’ aka ‘prototype’ instance by calling the **.SCHEMA.prototype()** method of **Person**. The second argument is the login name the user entered at the login page. We poll the Database with our new Query by calling the **.SCHEMA.findSingleAsset()** method, with our newly constructed **QueryEquals** as an argument. As the email of each user is unique, we will not find more than one Asset, and can therefore use the **find-**

SingleAsset() method, which returns null if nothing is found, or a **Person** instance if something is found.

We now have now retrieved our **Person** from the Database. If no one was found, or the password is wrong, we set a flag, and Return the '**FAILURE**' **Line**.

Failure and Failure Line:

It is here that we use the second type of **Station** a **Handler** can return as an argument: Pageful Handlers return **SimplePages**, pageless Handlers return **Lines**. It is a **Line** called '**Failure**' that we return here. A **Line** does not show a HTML page, like a **SimplePage**, instead, it forwards to another Handler:

```
final Line FAILURE = new Line() {
    @Override
    public void next(Forwarder f) {
        f.go(LoginHandler.class);
    }
}
```

As you can see, doing this is a bit more elaborate than in **SimplePage**, where the naming law tells Toro which HTML page to show. In **Line**, you must override a method called **next()**, which the framework will call when appropriate. **next()** expects a **Forwarder**, an object which Toro will supply, and which takes care of passing control on to the next Handler. The most obvious way to do this is by calling its **.go()** method, which you call with the class type of the Handler you wish to forward to.

Success and Success Line:

Of course, we hope that the user trying to log into our service is honest: In this case, we set the user via:

```
GenericSessionLocal.getSession().setUser(candidate);
```

In the future, we will always be able to find out whom the current session belongs to. As with any web-Service, many instances of our application can be running simultaneously, instantiated by different users, and knowing which user owns which session is important.

The '**SUCCESS**' **Line** is just like the '**FAILURE**' **Line**, with the obvious difference that it forwards to a different **Handler**.

3.12 Implementing Presentation – Print Substitutions

As you can see, failing to log in successfully means that control is once again passed to the **LoginHandler**. However, at this point, we would like to tell the user his mistake: Be it that he is not registered, or that he has entered the wrong password. We do this by displaying a certain String in the **login.htm** page. However, we want this String to

be flexible – depending on the error that occurred, we wish to replace this string with an appropriate message at runtime. In Toro, you mark a piece of HTML-code you will want to replace with a string at runtime with an opening and closing ‘\$’ sign, as in:

```
$this_is_a_PrintSubstitution$
```

Our Handler will then replace everything between the \$ and the \$ with a text at runtime.

First, let us insert our Print Substitution into the **login.htm** file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; char-
set=UTF-8"/>
    <script type="text/javascript" src="/__/jquery.js"></script>
    <script type="text/javascript" src="/__/platform.js"></script>
    <link rel="stylesheet" type="text/css" href="/__/Toro-
platform.css"/>
    <title>Toro Tutorial</title>
  </head>
  <body>
```

You must login....

```
$errorMessage$
```

```
<p>
  <form action="/submitLogin">
    Login: <input type="text" name="login"/>Password: <input
type="password" name="password"/>
    <input type="submit" value="Submit"/>
  </form>
</p>
</body>
```

This is the first thing we have to do. Now, we still have to supply the **LoginHandler** with a corresponding piece of code that takes care of replacing \$errorMessage\$ with an appropriate String. Here is the new Listing of **LoginHandler**:

```
public class LoginHandler extends Handler {
    private String errorMessage;

    @Override
    public void getParameters(ParameterReader parameters) {
        errorMessage = parameters.getString("errorMessage");
    }

    final SimplePage LOGINPAGE = new SimplePage() {
        @Override
        public void putSubstitutions(Template template) {
            template.put("errorMessage", new PrintSubstitution());
        }
    };
}
```



PrintSubstitution

```

        @Override
        public String print() {
            return Handler.notNull(errorMessage);
        }
    });
};

@Override
public Station doBusinessLogic() throws Exception {
    return LOGINPAGE;
}
}

```

In `getParameters()`, we retrieve the Error-Message passed to the `LoginHandler` from `SubmitLoginHandler`. (We haven't implemented this functionality in the `SubmitLoginHandler` just yet – we will do so right away).

PrintSubstitution:

A `SimplePage` which wants to substitute placeholders with text needs to override the `putSubstitutions()` method. When rendering the page, Toro will call this method and pass an argument of type `Template`. This object, in essence, is the webpage associated with the Handler through the naming law. By calling the `.put()` method of the `Template` object, we replace a specific placeholder – named in the first `.put()` argument through a String, here “errorMessage” – with a Substitution, specified through an instance of `PrintSubstitution` in the second `.put()` argument. This seems verbose, but the template-engine should take care of most of the typing: Just indicate what you want by typing a couple of letters and then press [ctrl]+[space].

Excursion:

“Verbosity, Conciseness, Clarity and Obscurity”

A common complaint about the Toro framework is that it is *verbose*. That is, it uses many words and symbols to accomplish its task. The Anonymous Inner Class mechanism, and method overriding, tend to be lengthy. For example, telling a `StringProperty` that its Maximum length is 100 involves the following code:

```

public final StringProperty country = new StringProperty() {

    @Override
    public int getMaxLength() {
        return 100;
    }

};

```

In classic Java, the same functionality might look something like this:

```
country = new StringProperty (maxLength=100);
```

Verbosity has two potential disadvantages: It can be hard to write, and it can be hard to read. Let us consider the first criticism:

Verbose code is hard to write.

Why? Because you need to type a lot of text. However, modern IDEs have code-completion systems that ‘rubberstamp’ text-blocks into your source-code when you only hint at what you want to do. Something that took hundreds of keystrokes to accomplish in earlier times can now be done in a few dozen keystrokes by using auto-completion. Toro-specific substitution templates are available. So verbose code no longer has to be hard to write.

Verbose code is hard to read.

This does not have to be true. In general, writing more code – like writing more documentation – should make code easier to read. In reading code, *clarity* is of essence. More code can mean more clarity, just as more words in a text can make an idea clearer. The opposite of Verbosity, Conciseness, can, in many cases, make code obscure: Humans have a tough time figuring out what the code does, even if a computer does not. But even computers can have problems interpreting concise code. They may be able to compile it, but refactoring and tooling are clumsy and error-prone. A verbose structure can help an IDE or framework to make sense out of the code, and enable meaningful highlighting of errors, reliable refactoring and the use of tools that help to view the whole project from a more abstract perspective.

As is typical of Toro, an instance of **PrintSubstitution** is AIC’ed on the spot. The substitution String will be generated at run time inside of the **.print()** method, which you need to override inside of the **PrintSubstitution** AIC. As a return value, **.print()** expects the String which will replace the placeholder in the HTML page. Here, we simply return the error message that was passed to us from **LoginHandler**. In case none was passed, the value of ‘*errorMessage*’ will be null. The static **.NotNull()** method of **Handler** will take care of this by replacing a possible null value with the empty-String.

TIP: Toro provides you with a useful tool to verify if you have implemented Substitutions for all HTML Placeholders in your Handler, the ‘Consistency Checker’. To show this tool, select: Window->Show View->Other->IMF->Consistency, and drag the appearing tab to a location comfortable to you.

To now find out if a Substitution is missing, open the Handler in the ‘Edit’ Window and make sure it is selected. Then, choose the ‘Consistency’ Tab from the Views. At this point, checking Consistency will not give any negative results, so let us sabotage the LoginHandler by replacing:

```
template.put("errorMessage", new PrintSubstitution() {
```

```

@Override
public String print() {
    return Handler.notNull(errorMessage);
}
});

```

with:

```

template.put("XXXErrorMessage", new PrintSubstitution() {
@Override
public String print() {
    return Handler.notNull(errorMessage);
}
});

```

At this point, **LoginHandler** is providing a **PrintSubstitution** for a Placeholder called "\$XXXErrorMessage\$" – which doesn't exist. This will do no harm, but not providing a **PrintSubstitution** for "\$errorMessage\$" will, and it is for this error we have introduced that we want to check. Run the Consistency Checker by clicking on the check-mark symbol in the right-hand top corner of the Consistency Checker:

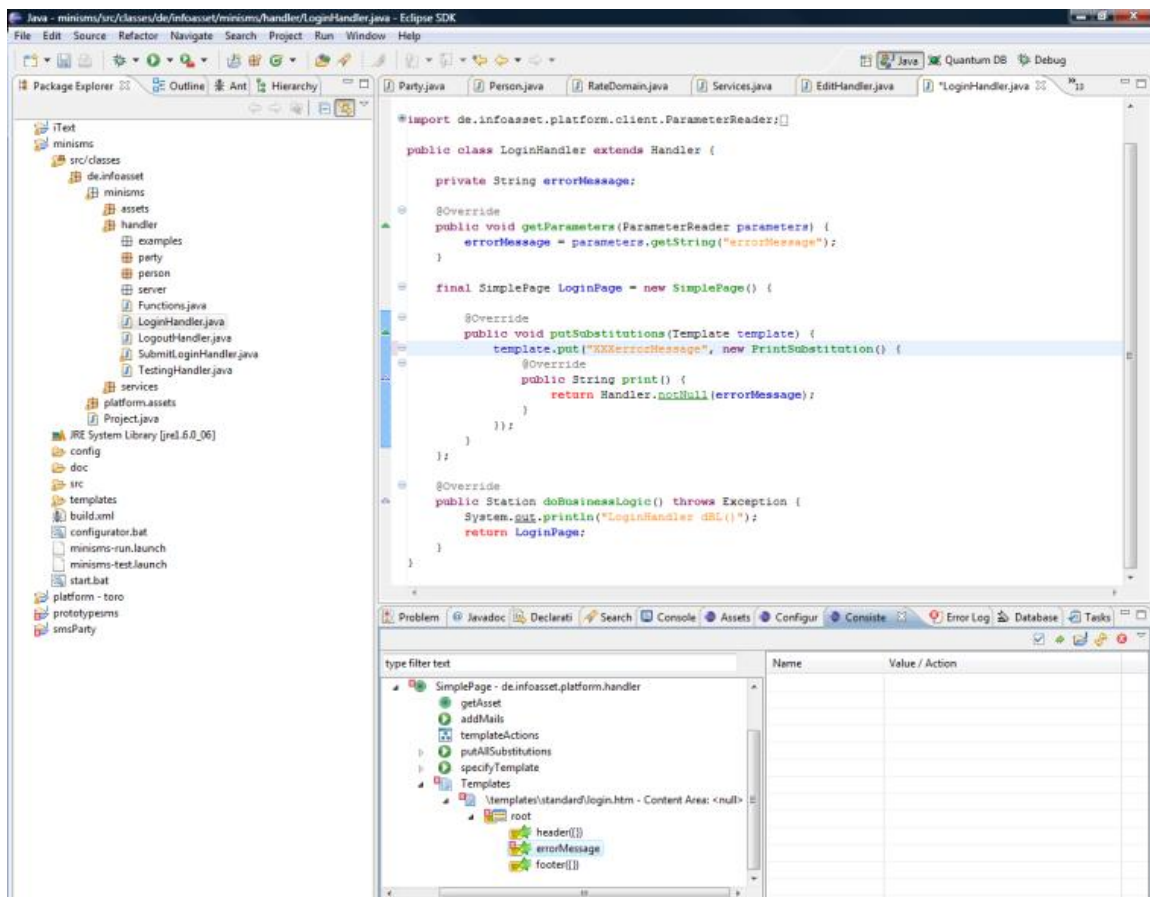


Figure 33: Checking for Consistency

Clicking your way through the Error Decorator popping up in the Consistency Checker (or clicking on the Error-Decorator Symbol in the right-hand top corner of the Consistency Checker) will tell you what is missing: A **PrintSubstitution** for "\$errorMessage\$".

3.13 Implementing Logic – Setting Parameters

What remains to be done is to ensure that appropriate Error-Messages are actually passed on. We do this by modifying **SubmitLoginHandler**:

```
public class SubmitLoginHandler extends Handler {

    private String login;

    private String password;

    @Override
    public void getParameters(ParameterReader parameters) {
        login = parameters.getString("login");
        password = parameters.getString("password");
    }

    final Line SUCCESS = new Line() {

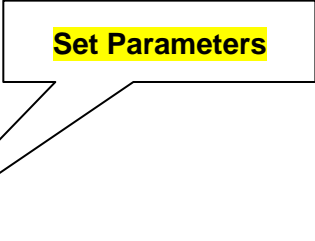
        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    final Line FAILURE = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(LoginHandler.class);
        }

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setMessage("errorMessage", noLogin, new Message() {
                String en = "Login missing.";
            });
            parameters.setMessage("errorMessage", notRegistered, new Message() {
                String en = "You entered an unknown e-mail address. Please try another e-mail address.";
            });
            parameters.setMessage("errorMessage", pwdError, new Message() {
                String en = "You entered a wrong password. Please try again.";
            });
        }
    };

    private Person candidate;
}
```



```

private boolean noLogin;

private boolean notRegistered;

private boolean pwdError;

@Override
public Station doBusinessLogic() throws Exception {
    if (login == null || login.length() == 0) {
        noLogin = true;
        return FAILURE;
    }
    candidate = Person.SCHEMA.findSingleAsset(new QueryE-
quals(Person.SCHEMA.prototype().eMail, login));
    if (candidate == null) {
        notRegistered = true;
        return FAILURE;
    }
    if (!candidate.password.get().equals(password)) {
        pwdError = true;
        return FAILURE;
    }
    GenericSessionLocal.getSession().setUser(candidate);
    return SUCCESS;
}
}

```

Set Parameters:

Every **Line** can have a **.setParameters()** method. It is here that the Parameters are set in the URL which forwards to the next **Handler**, which can then read them out with **.getParameters()**. Toro will pass an instance of **ParameterWriter** to **.setParameters()**. It is this object in which we write the parameters that should be passed via **.setMessage()**. In **SubmitLoginHandler**, we have used the three-argument version of **.setMessage()** of **ParameterWriter**: The first argument, a String, indicates the name of the parameter to be passed (in our case, always “errorMessage”), the second argument is a Boolean which decides whether the parameter should be set at all. In case the Boolean is ‘true’, the parameter is set according to the **Message** object given in the third argument. We have encountered the **Message** class before: You AIC it on the spot, and overwrite Strings, one for every language. (In the Tutorial, we only write the English ‘en’ String).

The sequence in which the functions are overridden has no influence on the order in which they are executed: **.getParameters()** is executed first, then **.doBusinessLogic()**, and finally **.setParameters()**.

At this point, you have everything in place to login successfully. Try it: Start up ‘minisms’, open the Browser of your choice, and go to: <http://localhost:8083/login>

The login and password can be seen from the Test Data we entered in chapter (3.9) (Login: mymail@myserver.com, password: prettyplease).

3.14 Implementing Presentation – Substitution-Functions and Template Substitutions

In many web-Pages, certain elements return over and over again. The HTML overhead is a good example, or a menu bar at the top of each page: The Pages may change, but these elements stay the same. Instead of copying and pasting these chunks of HTML code each time, it is a good practice to rubberstamp them into your HTML code through **TemplateSubstitutions**. These work like **PrintSubstitutions**, only that they copy HTML files into their placeholders, not Strings. In this way, changes in these rubberstamped elements propagate through all pages, changes which would otherwise have to be manually corrected in each location. In order to place identical Substitutions in all Pages, we need a Substitution that does not belong to a specific Handler, but works for all Pages. This is where Substitution-Functions come into play. They are available to all Pages, and can also digest arguments passed to them (we will not explain the argument-passing mechanism in this tutorial).

What we want to do is to replace the header on top of every HTML page:

header.htm:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; char-
set=UTF-8"/>
    <script type="text/javascript" src="/_/jquery.js"></script>
    <script type="text/javascript" src="/_/platform.js"></script>
    <link rel="stylesheet" type="text/css" href="/_/Toro-
platform.css"/>
    <title>Toro Tutorial</title>
  </head>
</body>
```

with a simple:

\$header()\$

The above Substitution-Function should then rubberstamp the content of **header.htm** into the web page.

So, first of all, we should create a HTML file called **header.htm** with the header code given above. This is a template, and must be placed in the `\minisms\templates\standard\` folder. As this template will be used in a Substitution Function, we should place it in an appropriately named folder, e.g. `\minisms\templates\standard\functions\`.

When Toro encounters a Print-Function Placeholder, i.e. something like **\$header()\$**, it looks for an identically named function in the **Functions** class, which is located in the `de.infoasset.minisms.handler` package. It is here that we have to supply the

header() function – a Java function which returns a Substitution for the placeholder `$header()`.

So let us put the following class **Functions.java** into the `de.infoasset.minisms.handler` package:

```
public class Functions {
    public static TemplateSubstitution header(FunctionParameters pa-
rams) {
        return new TemplateSubstitution() {

            @Override
            public void specifyTemplate(TemplateFinder templateName) {
                templateName.useStaticName("functions/header");
            }
        };
    }
}
```

As you can see, **.header()** returns a **TemplateSubstitution** (it could also have returned a different kind of Substitution, e.g. a **PrintSubstitution**, but we want to insert a piece of HTML code from a file here, so **TemplateSubstitution** is the way to go). The **TemplateSubstitution** is AIC'ed on the spot, and you override its **specifyTemplate** method. Toro will call this method when the time comes to replace the placeholder, and pass it an object of type **TemplateFinder**. **TemplateFinder** has several methods of finding its template – we **useStaticName()**, and specify the relative path of **header.htm**.

We can now proceed to replace the HTML overhead in all of our previously created pages with `$header()`. You should do so now, because we will be modifying the **header.htm** file, and do not want to do so for each page we create, but only in one place.

As an exercise, you should now try and replace the `</body>` tag at the end of each HTML page with a `$footer()` function.

3.15 Implementing Presentation – Conditional Substitutions

We often want to show information on a web-page only if a certain condition is met. For example, in our application, we would like to have a link at the top of the page referring to all of the User-Settings if a User is logged in. If no User is logged in, we do not want to display such a link. This functionality is supported in Toro through a type of Substitution called **ConditionalSubstitution**.

In the HTML code, we protect a region which we only want to display conditionally with the following Syntax:

```
[$conditionGuard$
Protected region – only displayed if condition is 'true'
$conditionGuard$
```


Everything between the `[$conditionGuard$` and the `[$conditionGuard]$` symbols is not rendered if the condition should prove to be false.

Which condition?

Just as with a **PrintSubstitution** placeholder such as `[$errorMessage$`, where the **Handler** associated with the web-Page in which the placeholder appears must provide a `.put()` with a **PrintSubstitution** (see chapter (3.12)), a condition-guard appearing in a web-Page must have a corresponding `.put()` providing a **ConditionalSubstitution**.

So let us assume we wanted the condition guard to always be true, then we would provide the following **ConditionalSubstitution** in the corresponding **Handler** of that web-page:

```
final SimplePage PAGE = new SimplePage() {

    @Override
    public void putSubstitutions(Template template) {
        template.put("conditionGuard", new ConditionalSubstitution() {

            @Override
            public boolean test() {
                return true;
            }
        });
    }
};
```

As you can see, the center-piece of a **ConditionalSubstitution** is the Boolean `test()` function, which you must override, and which implements your test-logic.

In our case, we will not be hooking into the `.putSubstitutions()` of a specific **Handler's SimplePage** to implement our **ConditionalSubstitution**: We want the conditional User-Link to appear in all of our web-pages, and will therefore use a Conditional Substitution-Function, available to all pages, defined in **Functions.java** in the `de.infoasset.minisms.handler` package:

```
public class Functions {
    public static TemplateSubstitution header(FunctionParameters params) {
        return new TemplateSubstitution() {

            @Override
            public void specifyTemplate(TemplateFinder templateName) {
                templateName.useStaticName("functions/header");
            }
        };
    }

    public static ConditionalSubstitution anonymous(FunctionParameters params) {
        return new ConditionalSubstitution() {
            public boolean test() {
```

Conditional Substitution

This looks slightly more complicated than the Conditional-Substitution placeholder you saw earlier. Actually, the syntax of Conditional-Substitution placeholders is richer than a simple on-off alternative:

```
[$conditionGuard$  
Protected region - only displayed if condition is 'true'  
$]conditionGuard[$  
Protected region - only displayed if condition is 'false'  
$conditionGuard]$
```

Applied to our example, this means that if the User is anonymous, i.e. not logged in, we will display a heading of 'Toro Tutorial'.

In case he is logged in, i.e. the 'anonymous' condition is false, we do not display the 'Toro Tutorial' heading, but instead two links, one linking to a page where he can edit his User Data, and one logging him out of the system. (We haven't provided Handlers or Pages for these two links yet – we will do so shortly.)

Inside of the link you can see two other Substitution-Functions being used:

```
$currentUserId() $ and $currentUserName() $
```

We just inserted `$currentUserName()$` into `Functions.java`, so you know what it does, but where does `$currentUserId()$` come from, which we use to pass an argument to the (still not implemented) `EditHandler`? In chapter (3.10), we mentioned the 'fall-back' project 'platform-toro, which also includes several Print-Functions, among them `$currentUserId()$`. If you haven't supplied a Print-Function appearing in a web-page, Toro will look to 'platform-toro, and only if it does not find it there will it complain. `$currentUserId()$` will be replaced with the id-Property value of the current User. Actually, an `$anonymous()$` Print-Function is also supplied by 'Toro-platform', so we did not really have to implement it ourselves – but we did so anyway, to learn about **ConditionalSubstitutions**.

3.16 Implementing Presentation – Editing-Forms for Assets

At this point, we can log into our application using the test-account. The data for this test-account was persisted in the Database in the `initData()` method of the `Services` class (see chapter (3.9)). However, how do we let the user edit this data?

In the previous chapter, we inserted a link which we want to point to a page in which this data can be edited. Building this page and its Handler will be the goal of this chapter.

The following State-Diagram shows where this functionality fits in:

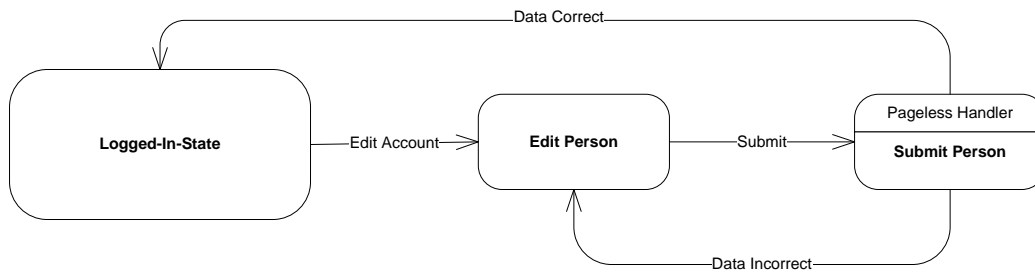


Figure 34: Editing an existing account

'Logged-In-State' is a super-state representing all pages that have the link which we placed into the `header.htm` with the Conditional Substitution.

Everything we do here is has to do with the **Person** Asset, so we will place the **EditHandler** into the `de.infoasset.minisms.handler.person` package:

```

public class EditHandler extends Handler {

    private String accountId;

    private Person account;

    @Override
    public void getParameters(ParameterReader parameters) {
        accountId = parameters.getString("id");
    }

    final SimplePage EDITPAGE = new SimplePage() {
        @Override
        public Asset getAsset() {
            return account;
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        account = Person.SCHEMA.getAsset(accountId);
        return EDITPAGE;
    }
}
  
```

Obtain the Id

Return the Asset

Obtain the Asset

The main job of **EditHandler** is to retrieve the correct **Person** object from the Database and supply this object to the web-page, which will then display it in a web form (using a new Substitution-Function we will explain next).

Obtain the Id:

If you look into the link to the **EditHandler** (remember: It might look like a link to a web-Page, but it actually links to a **Handler**), you see that a Parameter called 'id' is passed. This parameter is retrieved here. Every Asset has a special **Property** of type **IdProper-**

ty called *id*, which is a Toro-generated String uniquely identifying every instance of an Asset. You cannot (and should not!) **.set()** this *id* yourself, Toro does it for you, but it is a very convenient value to pass between **Handlers** via **.getParameters()** and **.setParameters()**.

TIP: How you name the parameter which identifies the Asset which is passed to the following Handler is, of course, completely up to you, there is now law, but a good convention is to simply call it *id* – and to not be any more verbose than this. From the context of the receiving Handler, it should be completely clear what kind of *id* is being passed – if it is a Handler dealing with **Party**, it will be a party-id. If it is a Handler dealing with **Person**, then it is a person-id. Not sticking to this convention can be a bother, because you will always have to refer to the link in the html file from which the parameter was passed in order to parse the correct parameter in **getParameters()**.

Obtain the Asset:

Here we retrieve the Asset with the **SCHEMA.getAsset()** of **Person**. **getAsset()** expects an *id*-value, and we use the one passed to this **Handler**. Passing a whole Asset directly between **Handlers** is not supported in Toro: You must always pass a unique identifier in the URL, and then retrieve the Asset from the DB using it. This unique identifier is usually the *id*, but does not necessarily have to be. You can choose any key-value of your Asset.

Return the Asset:

Here we encounter a new function of **SimplePage** which we may override if we wish to: **getAsset()**. In it, we simply return a single Asset which we want to put at the disposal of the web-page. We return our freshly retrieved *account*, the Asset passed to us by the previous **Handler**.

The associated HTML page associated with this Handler is very concise, and almost self explanatory. You must call it **edit.htm** and place it in the `\minisms\templates\standard\person\` folder:

```
$header() $  
  
<form method="post" action="/person/submit?id=$id.value()" $"  
class="Toro-edit-form">  
    $edit(exclude=isAdministrator, membershipDate) $  
</form>  
  
$footer() $
```

Before further commenting on this HTML code, let us run 'minisms', log in, and then click on the link (mymail@myserver.com) at the top guiding us to the web-page in question:

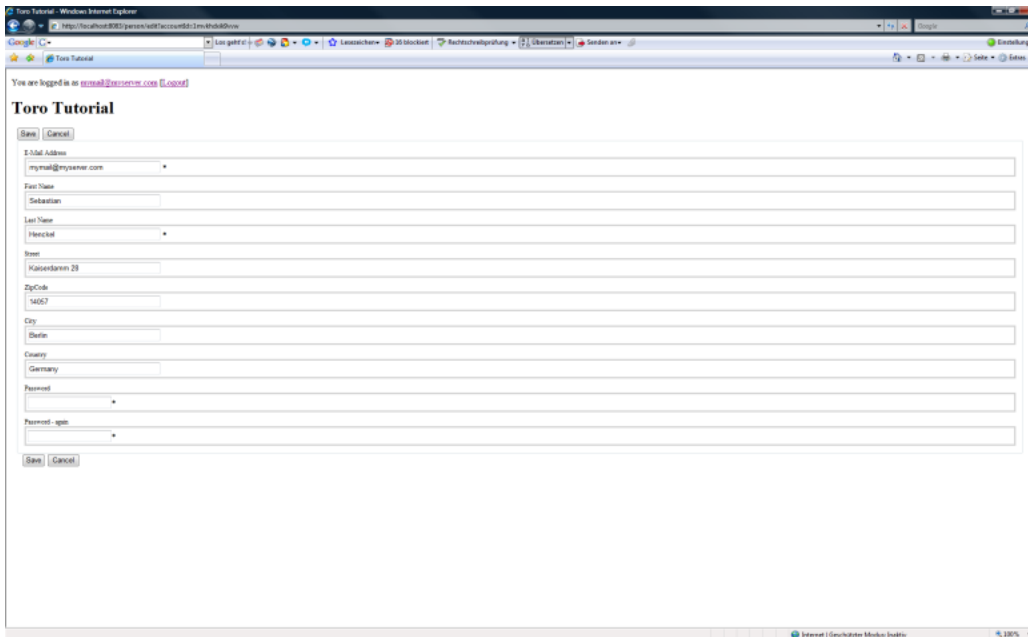


Figure 35: The Edit-Page of the Person Asset

As you can see, the `$edit(exclude=isAdministrator, membershipDate)$` Substitution-Function does a lot of work: For every **Property** in the Asset we passed to its **SimplePage**, `$edit()` will display an edit-field preceded by the Label we gave that **Property** (look back to chapter (3.4) to see how we labeled Properties). Also, it displays the current values of the Asset in the edit-fields. Properties we flagged as mandatory with a **NotNullValidator** are decorated with a *. The **PasswordProperty** is dealt with appropriately in a fully automatic way: It is not displayed, and two edit-fields make sure that it is typed in correctly.

As arguments to `$edit()`, we exclude the **Properties** `'isAdministrator'` and `'membershipDate'`, which are not displayed. This makes sense: We do not use the `'isAdministrator'` functionality in this tutorial-application, and `'membershipDate'` is a value that is set by the system, not the user.

Two short notes on the HTML form used here: Toro depends on certain `.css` classes to display its forms correctly, and you must assign `class="Toro-edit-form"` in the form tag. The action this form takes is to link to a **submitHandler** which we have not written yet – we will do so shortly. As an argument, we pass a parameter `'id'`, which we assign the `'id'` of the Asset associated with this page via `$id.value()`: All **Property** values of an Asset associated with a Page can be displayed with the `$.value()` Substitution function – you just have to place the name of the Property you wish to display before the `'dot'`, e.g. `$id.value()` or `$email.value()`.

Let us now write the **submitHandler** for the **Person** Asset. We have placed this **Handler** in the `de.infoasset.minisms.handler.person` package, following our convention of placing all **Handlers** dealing with an Asset into a dedicated package:

```

public class SubmitHandler extends Handler {

    private String accountId;

    private Person newAccount;

    @Override
    public void getParameters(ParameterReader parameters) {
        accountId = parameters.getString("id");
    }

    final Line VALID = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    Line INVALID = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newAccount.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        newAccount = Person.SCHEMA.getAsset(accountId);
        newAccount.makeTransient();
        newAccount.applyParameters();
        if (newAccount.isValid()) {
            newAccount.makePersistent();
            GenericSessionLocal.getSession().setUser(newAccount);
            return VALID;
        } else {
            return INVALID;
        }
    }
}

```

Checking Validity

There is little new here: It is a pageless **Handler** with two **Lines**, the '**VALID**' **Line** being returned by **doBusinessLogic()** in case the Asset turns out to comply to all **Validators**, the '**INVALID**' **Line** is returned in case not. The '**INVALID**' **Line** bounces the user back to the Edit web-Page, passing the '**id**' of the invalid Asset back to the **EditHandler**. The **\$edit()** Function-Substitution of the Edit-Person web-page will then take care of highlighting the faulty Properties, and displaying the error Messages that are associated with the offended **Validators**.

It is in the **doBusiness()** method that something new appears:

Checking Validity:

With the **.makeTransient()** method of an Asset, we (temporarily) allow an Asset to take on inconsistent values in its Properties, i.e. values offending the attached **Validators**.

Once we have verified the Asset's consistency, we can then persist it with **.makePersistent()**. Checking for consistency is done with the **.isValid()** method of an Asset, which checks for compliance with all **Validators**.

After persisting the changes in **Person**, we set the Session-User once again with:

```
GenericSessionLocal.getSession() .setUser(newAccount);
```

This is superfluous if we only changed the **Person** data, but we must also account for the case in which we arrive at this **SubmitHandler** from a newly created account! We have been logging into the application through the dummy-account we set up for testing purposes. Now we should also provide a functionality which allows the user to set up his own, new account. In order to do this, let us provide a link to this functionality in our **login.htm**:

```
$header()$
```

```
You must login....
```

```
$errorMessage$
```

```
<p>
  <form action="/submitLogin">
    Login: <input type="text" name="login"/>Password: <input
type="password" name="password"/>
    <input type="submit" value="Submit"/>
  </form>
  [
```

```
$footer()$
```

As you can see, I assume you have replaced the leading and trailing HTML code with Substitution-Functions.

The link we have added to the **login.htm** page leads us to a pageless **Handler** called **NewHandler**. Place it in the `de.infoasset.minisms.handler.person` package:

```
public class NewHandler extends Handler {

    final Line EDIT = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newAccount.id.get());
        }
    }
}
```



```

@Override
public void next(Forwarder f) {
    f.go(EditHandler.class);
    f.noRedirect();
}
};

private Person newAccount;

@Override
public Station doBusinessLogic() throws Exception {
    newAccount = Person.SCHEMA.createTransientAsset();
    return EDIT;
}
}

```

The only point of **NewHandler** is to create a new **Person** which we can then pass to the **EditHandler**.

Creating a transient Asset:

The **.createTransientAsset()** method of an Asset's 'SCHEMA' is used to supply a transient Asset. In chapter (3.9) we encountered **.createAsset()**. The difference between a transient and a normal Asset lies in the fact that a transient Asset is not persisted until explicitly told to do so with the Asset's **.makePersistent()** method. A normal Asset is persisted automatically once the creating **Handler** is left, and the ensuing **Handler** is called. In case of the **NewHandler**, the newly created Asset '*newAccount*' is empty (and therefore inconsistent), and we do not want to persist it when crossing **Handler** borders. Making it persistent will be left to the **SubmitHandler**.

No Redirection:

The **.noRedirect()** method of the **Forwarder** is invoked in case you do not want the browser's URL field to display the URL of the resource the **Forwarder** is heading to. In our case, we will end up on the <http://localhost:8083/person/edit> page, but the browser will still display the URL of the preceding **Handler**, i.e. <http://localhost:8083/person/new>. This makes more sense – the difference is rather cosmetic.

3.17 Implementing Logic – Queries on the Database

In chapter (3.11) we introduced a dummy page and Handler `\minisms\templates\standard\party\list.htm`, and the corresponding **ListHandler**. Now, we want to fill these dummies with life. After a successful login, this page should display a complete list of all the parties belonging to the user. In a first step, we must retrieve all parties associated with the user who logged in, and who now

owns the session. In the next chapter, we will show how to render this list with Toro using a special kind of substitution – the List-substitution.

In chapter (3.11) we already learned how to find a single Asset whose Property has a certain value- in order to check for a valid login. Now, we find the **Person** Asset of the current user, and then JOIN it with the **Party** Assets. Finally, we project the Result of this JOIN to the **Party** values.

Excursion :

“SQL vs. Toro”

The Query described above corresponds to the SQL command:

```
select Party.*  
from Party,Person  
where Party.organizer = Person.id AND Person.id = 66
```

Java does not support SQL natively. You can access a Database in Java through SQL using JDBC. JDBC sends SQL Strings to a JDBC driver, which connects to the Database. This approach is not type-safe and does not support any kind of error-checking at edit- or compile time: Mistyped SQL statements, supplying Integer compare values for String columns – all this will not be caught before you run the program. You are just sending command Strings to a JDBC driver. If they actually compute can only be seen when the application tries to execute them. Funneling the Query-Results into Java is done through an Iterator-like **ResultSet**, a JDBC specific class.

A better alternative is SQLJ, which at least offers compile-time error-checking: SQLJ relies on escaped SQL statements and Java-like Iterators. However, you need an additional compiler for these before you use the Java compiler on your code [AE01]. Also, both SQLJ and JDBC do not abstract from the different SQL dialects, and error-decoration at edit-time, one of Eclipse’s highlights, is not supported by either of them.

The Toro Query language, on the other hand, is easy to learn, type-safe, and checks for errors during edit-time. It is Java, not SQL, and delivers its results to a simple **Iterator<>** on an Asset.

In order to find all Parties which are owned by the user owning the current session, we write:

```
QueryEquals q = new QueryE-  
quals(Party.SCHEMA.prototype().organizer.getAttributeSignature(), Gener  
icSessionLocal.getSession().getUser().id.get());
```

What we are doing here is referencing the **OneRole** ‘organizer’, which holds the Foreign-Key to the **Person** objects. Obtaining this key is done with the **.getAttributeSignature()** method. We then compare this value to the key-value of the current user (which we obtain via **id.get()**). **QueryEquals** assures these two values match – that is, in executing this **Query**, we will obtain all **Assets** where the **Party** objects belong to the current User.

QueryEquals is just one of several **Query** Classes that Toro offers, such as **QueryBegins**, **QueryContains**, **QueryGreater** etc. Check the Class hierarchy of **Query** to see all derived Classes.

Of course, we can also combine two Queries into a new **Query** which fulfills both conditions: This would be done with **QueryAnd** Class, using the **QueryAnd(q1,q2)** Constructor, which takes two Queries as arguments that should be linked with AND. OR is also possible, with the **QueryOr** Class, which works the same way.

After having applied our Queries to our Assets, and stacked them together with ANDs and ORs, we specify in which way the results should be ordered:

```
q.addSortingCriterion(new SortingCriterion(Party.SCHEMA.prototype().begin, SortingCriterion.ASCENDING));
```

.addSortingCriterion() takes a **SortingCriterion**-object. You should create it on the spot, passing as first argument to its Constructor the **Property** you wish to order by, and as the second argument the static Boolean constants of **SortingCriterion**, 'ASCENDING' or 'DESCENDING'.

Lastly, we execute the **Query**:

```
Iterator<Party> myParties = Party.SCHEMA.queryAssets(q);
```

Choosing which set of **Property**s from which **Asset** the Query-Result is projected upon is done by calling **.queryAssets()** from the **.SCHEMA** of the **Asset** we wish to obtain. (This corresponds to the 'SELECT * FROM' Statement in SQL). The Query-Result is returned in form of an **Iterator<>** Generic, which we specialize to the **Asset** the **Query** returns, in this case **Party**.

We could pack the above Query in **doBusinessLogic()**, but this would mean abusing the actual calling of **doBusinessLogic()**, which should only manipulate Data and decide which **Station** to return, not retrieve objects from the Database for presentation-purposes. (Not placing anything but Data-manipulation and Decision-making code in **doBusinessLogic()** is a convention, not a law). We will see where to appropriately place this code in the next chapter.

3.18 Implementing Presentation – ListSubstitutions

We have no way of knowing beforehand how long our list will be, and even if we did, it would be very repetitive to define a **PrintSubstitution** for every element in our list. This is where **ListSubstitutions** come in handy. A List Substitution, like all Substitutions, consists of two parts: One is a template-placeholder that goes in the HTML page, and the other is code that substitutes that placeholder at run-time. Let us put the placeholder code into `\minisms\templates\standard\party\list.htm`:

```
$header()$
```

```
<h1>"Parties" Page of $user$</h1>
```

PrintSubstitution

```
<table>
  <tr>
    <th>
      Title
    </th>
    <th>
      Location
    </th>
    <th>
      Begin
    </th>
    <th>
      Price
    </th>
```

Table Heading

```
    <th>
      Purchase Date
    </th>
  </tr>
```

```
  $[parties p$
  <tr>
    <td>
      <a
href="/party/view?id=$p.id.value() $">$p.name.value() $</a>
```

List Substitution

```
    <td>
      $p.location.value() $
    </td>
    <td>
      $p.begin.value() $
    </td>
    <td>
      $p.price.value() $
    </td>
```

```
    <td>
      $p.purchaseDate.value() $
    </td>
  </tr>
```

New Party Link

```
  $parties]$
</table>
```

```
[<a href="/party/new">Create a new Party</a>]
```

```
$footer()$
```

Print Substitution:

Nothing new here. We want to display the user's login (=email) at the top of the page. We will put the corresponding **PrintSubstitution** into **ListHandler** in a moment.

New Party Link:

We put a link to a (not yet existent) **Handler** at the bottom of the page which will take care of creating new Parties.

Table Heading:

This has nothing to do with Toro: We are simply opening a new table and establish four columns which we call 'Title', 'Location', 'Begin' and 'Purchase Date'. Below this row we want to start displaying our **PartyAssets** we have retrieved from the Database.

List Substitution:

It is here that we have something new. We open and close our list with the following placeholder:

```


 $\$[listname\ l\$$ 
Iterator-Domain. Use Iterator 'l' to access current Asset in the successive rows of the table.
 $\$listname]\$$ 


```

A list-placeholder consists of a listname and an Iterator symbol (,listname' and ,l', respectively, in the above example, and ,parties' and ,p', respectively in **list.htm**.)

Everything between and including these placeholders will be replaced at run time by a repeating block of HTML code, which has the same structure as the HTML code in place: What Toro does is to repeat the code block inside of the placeholder-guards as many times as there are **Assets** supplied by **ListHandler**, and replacing terms like `$p.name.value()` with the value of the indicated **Property** of the **Asset** returned in the current Iteration. `value()`, is an inbuilt Print Substitution Function, which displays the value of a **Property**.

What we still need is to implement the mechanism inside of **ListHandler** which takes care of supplying and progressing through the list of **Assets**. Let us do so now. Insert the highlighted code into **ListHandler** in the `de.infoasset.minisms.handler.party` package:

```

public class ListHandler extends Handler {
    private Person user = GenericSessionLocal.getSession().getUser();

    private Iterator<Party> myParties;

    final SimplePage LIST_PAGE = new SimplePage() {
        @Override
        public void putSubstitutions(Template template) {
            template.put("parties", new ListSubstitution() {
                Party currentParty;
                Iterator<Party> myParties;

                @Override
                public void start() {

```

Put Substitutions

ListSubstitution

```

        QueryEquals q = new QueryE-
quals(Party.SCHEMA.prototype().organizer.getAttributeSignature(), us-
er.id.get());
        q.addSortingCriterion(new SortingCrite-
rion(Party.SCHEMA.prototype().begin, SortingCriterion.ASCENDING));
        myParties = Party.SCHEMA.queryAssets(q);
    }

    @Override
    public void next() {
        currentParty = myParties.next();
    }

    @Override
    public Asset getCurrentAsset() {
        return currentParty;
    }

    @Override
    public boolean hasNext() {
        return myParties.hasNext();
    }
});
template.put("user", new PrintSubstitution() {
    @Override
    protected String print() {
        return user.eMail.get();
    }
});
};

@Override
public Station doBusinessLogic() throws Exception {
    return LIST_PAGE;
}
}

```



Put Substitutions:

At the start of **SimplePage**, we have now inserted the **.putSubstitutions()** method.

ListSubstitution:

It is here that we supply everything we need to substitute the placeholder in our HTML file:

start() : In this method, you retrieve the data you want to render in your list. It is here that we query the Database in the way we learnt in the previous chapter.

next() : Here we update our Asset-reference to the new Asset.

getCurrentAsset : We return the (updated) **Asset**-reference.

hasNext() : A Boolean Function telling us whether the list still has elements. We have to override it appropriately.

PrintSubstitution:

In it we provide the 'user' placeholder with its **PrintSubstitution**. This should be familiar from chapter (3.12).

If you now log into the application, you will see a single **Party** listed – the one we entered as test-data in **Services**. If we want to see more **Partys** we will have to add them ourselves. So let us now implement the functionality of adding a new **Party**. This is nothing new, as we will proceed in the same way we did when creating a new user-account. But in order to make things interesting we will determine the price of the new **Party** using a “configurable” price.

3.19 Implementing Logic – Configuration

Besides users and programmers, applications usually encounter a third group of people, administrators. Changing the functionality of an application may require the programmer to adapt the source-code, but often, the change needed is superficial and should not require source-code modification. Implementing these changes is the job of the administrator.

In our example, fixing rates and prices of new parties should be done by an administrator, not the programmer, who should not be bothered after his job is done, (and the user should certainly not be able to do it!).

However, the programmer should devise an interface that allows the administrator to change these settings for the application.

One possibility would be to build certain web-pages that are only accessible to the administrator, where he can configure the application. Toro, however, has a separate configuration mechanism that is independent of HTML and the underlying Database.

The configuration of the tutorial application are saved in a .xml file called **configuration.xml** located in `\minisms\config\`.

It is this file which you pass as a command-line argument to the **main()** method of the framework. Everything that happens thereafter is determined by **configuration.xml**: What Database and JDBC Driver should be used? What language should the application be displayed in? Yes, even which application the framework should launch is set here.

In `\minisms\config\` you will find another .xml file called **repository.xml**. This file describes all configurable traits of the application – not the configuration itself - that is done by **configuration.xml**.

Let us configure our new application a bit – for example, at this moment, we are prompted at the start of our application to confirm that we want to delete all data in our Database. This is useful for testing purposes, but once debugging is over, we want to

retain our data between launches. This is something we can configure, and let us do so now:

First, we have to launch the 'Configurator'. This a tool provided by Toro to let you configure your application. Launching the Toro-Configurator is done through an Ant-Target. In case you have not done so already, turn on the Ant-View Tab, and add the **build.xml** Ant-file –located in the 'minisms' project- to your build files in the Ant-View (Select Window->Show View->Ant, click on the Ant-Tab, and click on the 'Add Ant build File' icon).

Launch the 'Configure[default]' of the 'minisms' build-file target in the Ant-View by double-clicking on it. The Configurator appears:

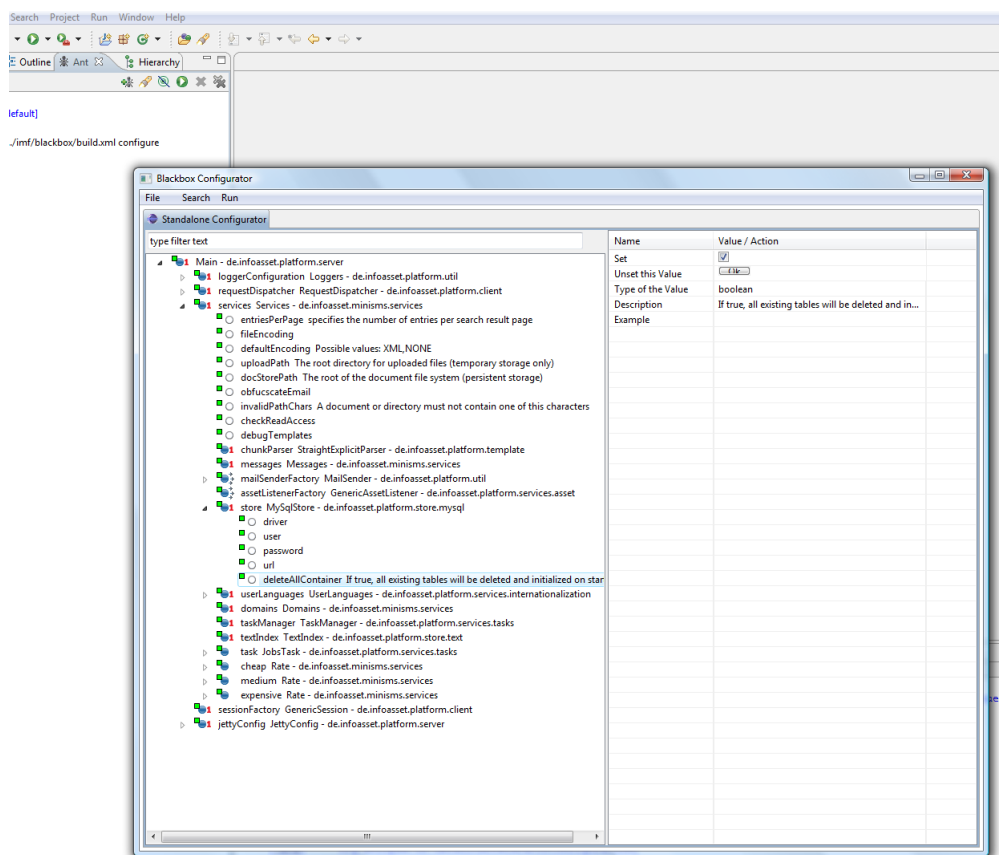


Figure 36: Configuring a setting in the application

As you can see, there are very many settings you can change in your software. Most of them are rarely needed, and should stay the way they are. The switches are ordered hierarchically. We will comment on this hierarchy shortly, but at the moment, we are interested in only one switch, called 'deleteAllContainer'. If you know the name of a switch, finding it is best done with the search-field at the top of the Configurator. Type in 'deleteAllContainer', and only the specified switch will remain in the Configurator. To change the value, uncheck the 'Set' box, and then choose 'File->Save'. (Do not click

the 'Unset this value' button – doing so does not turn the setting off, but rather leaves it undefined. The decorator will turn red, indicating that you have an incomplete configuration). When you restart the application, you will no longer get the warning about the repository being deleted, and the Database will no longer be flushed. (For the rest of this tutorial, we will assume that 'deleteAllContainer' is set, however, so you should set this switch again and save the configuration).

Now that we have seen how to configure a Toro-application, let us learn how to create our own configurable settings in our application.

Configurable elements in Toro are installed into the application using Annotations, a feature added to Java in the [Ec06].

We would like to have three configurable instances of a class called **Rate** ('cheap', 'medium' and 'expensive'):

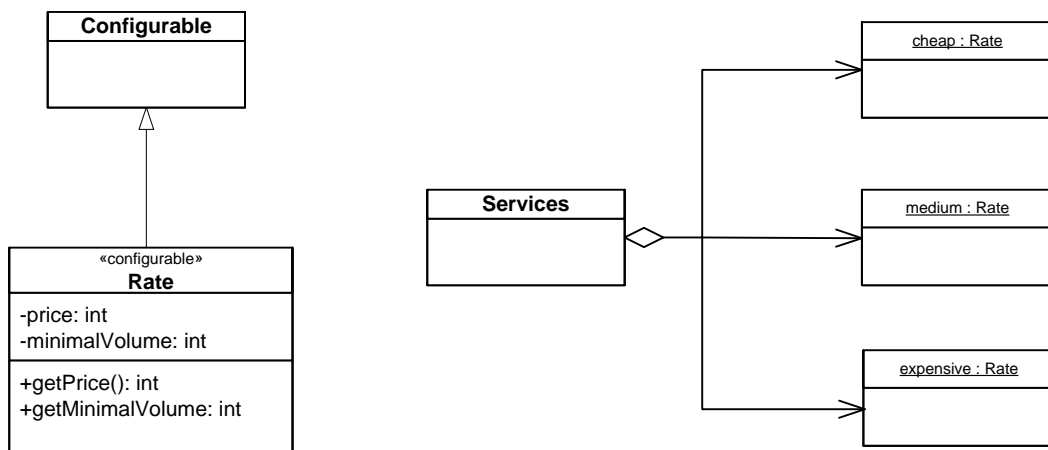


Figure 37: Class Diagram of **Rate** and its instances

The two int-attributes of **Rate**, 'price' and 'minimalVolume', represent the amount of cents per guest a customer has to pay at that rate, and, respectively, at what number of purchased parties this price sets in (the more you buy, the cheaper it gets). We can make these attributes 'private', because we don't want any Class (actually, not even the *owning* Class, but this is impossible) to change these values: They should only be modifiable through the Configurator, where an Administrator can decide how prices should go down with increasing purchase-volumes.

Once we have the new **Rate** class, we will put three instances in a general-purpose class, e.g. **Services**. All of this is shown in Figure 37).

First, let us create our new configurable **Rate** class. We place it in the `de.infoasset.minisms.services` package, the same package **Services** is in, the class that will contain the future three instances of it:

```
public class Rate implements Configurable {
    @Property
    private int price;

    @Property
    private int minimalVolume;

    public int getMinimalVolume() {
        return minimalVolume;
    }

    public int getPrice() {
        return price;
    }
}
```

Configurable Class:

Any Class which contains configurable Attributes (marked with the **@Property** Annotation, see below) must implement the **Configurable** interface.

Configurable Attribute:

Any Attribute that you wish to make configurable must be marked with the **@Property** Annotation. In its current version, Toro only supports configurability for primitive types.

Let us now add three instances of **Rate** – one for each pricing-scheme- to the **Services** class (located in the `de.infoasset.minisms.services` package):

```
public class Services extends GenericServices {
    private static Services instance;

    public static Services INSTANCE() {
        return instance;
    }

    @Association
    private Rate cheap;

    @Association
    private Rate medium;

    @Association
    private Rate expensive;

    public Rate getCheap() {
        return cheap;
    }

    public Rate getMedium() {
```

```

        return medium;
    }

    public Rate getExpensive() {
        return expensive;
    }

    protected void initData() {
        .
        .
    }
}
<snip>
    }
    protected void initSchemas() {
        initSchema(Person.class);
        initSchema(Party.class);
    }
}

```

Setting them:

We have annotated the three instances of our **Configurable** class **Rate** with the **@Association** annotation. This tells the Toro-framework that the Configurator is responsible for setting these values. Simply making a class a **Configurable** is a necessary, but not a sufficient condition for it to be configured via the Configurator: You might want to use instances of **Rate** for other purposes, and these instances should not be set by the the Configurator tool. You would then omit the **@Association** annotation in front of these instances (we won't be using any other instances of **Rates** here, but having certain instances of a class that should be set through an administrator, and other instances that you the programmer want to set is a possibility).

The Getters:

These are boiler-plate getters we will use to access the attributes.

It is now possible to configure our new rate-objects – almost: The Configurator hasn't yet noticed the three new **Configurable Rates**, as it still has an old version of **repository.xml**. We must add the new **Configurables** to **repository.xml**. Scanning the source-code for new Configurables is done through a plug-in tool inside of Eclipse, the Configurator-tab. (This is also called a Configurator, but is distinct from the Stand-Alone Configurator). If the Configurator-tab does not show in your Eclipse IDE, you must turn it on by choosing Window->Show View->Other... and then selecting 'Configurator' in the IMF folder. Now click on the Configurator tab, and click on the 'Build the

repository for this project' icon (a folder with a J on top of it). You are prompted to choose the project for which you want to update the `repository.xml`. Choose 'minisms' (Figure 38)).

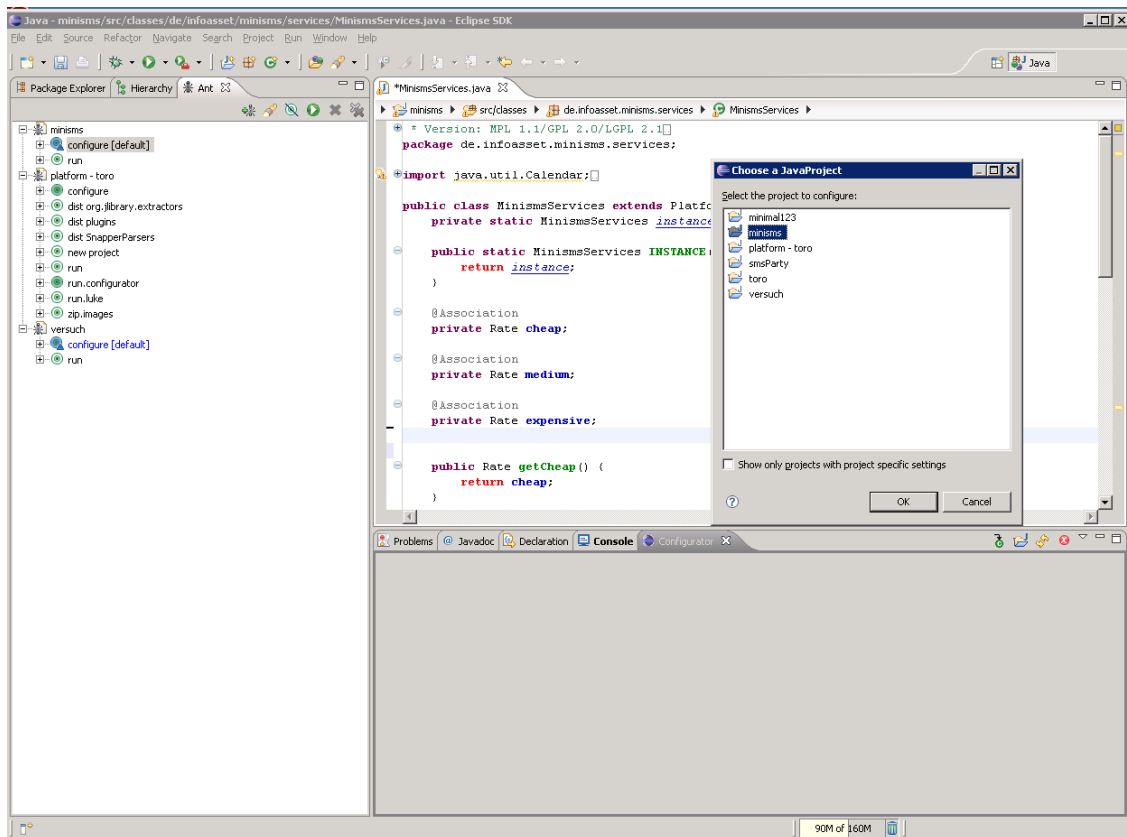


Figure 38: Telling Toro to search for new switches

Tip: The process of building a new configuration needs a lot of memory, and in case you keep getting error messages, try clicking on the 'release all Resources' icon in the Configurator-tab, which frees unused memory.

Now close and reopen the Stand-alone Configurator, and you should find the three new configurable instances of **Rate** (under 'services'). However, you still can't set the attributes of the new **Rates**! The problem is that configuration in Toro can also be done on the level of a class: Not only attributes of a class can be set, you can also set the class itself. For example, we could have introduced two subclasses of **Rates**, **CoroprateRate** and **PrivateRate**, with extra fields for each variety of class, such as an extra Corporate Rebate Percentage for **CoroprateRate** and a Boolean 'special offer'-switch for **PrivateRate**. You could then change the instance of a class with the Configurator from **CoroprateRate** to **PrivateRate** without touching the code (in the code, you would have to be instantiating the mother-class, **Rate**, so that the reference can hold both types). Class level configuration is an advanced feature, and we will not go into it. In this tutorial, we only have one class-type, but the Configurator still wants us to choose

it. So click successively on the three **Rates**, select the **Rate** class, and click the 'OK' button (Figure 39)).

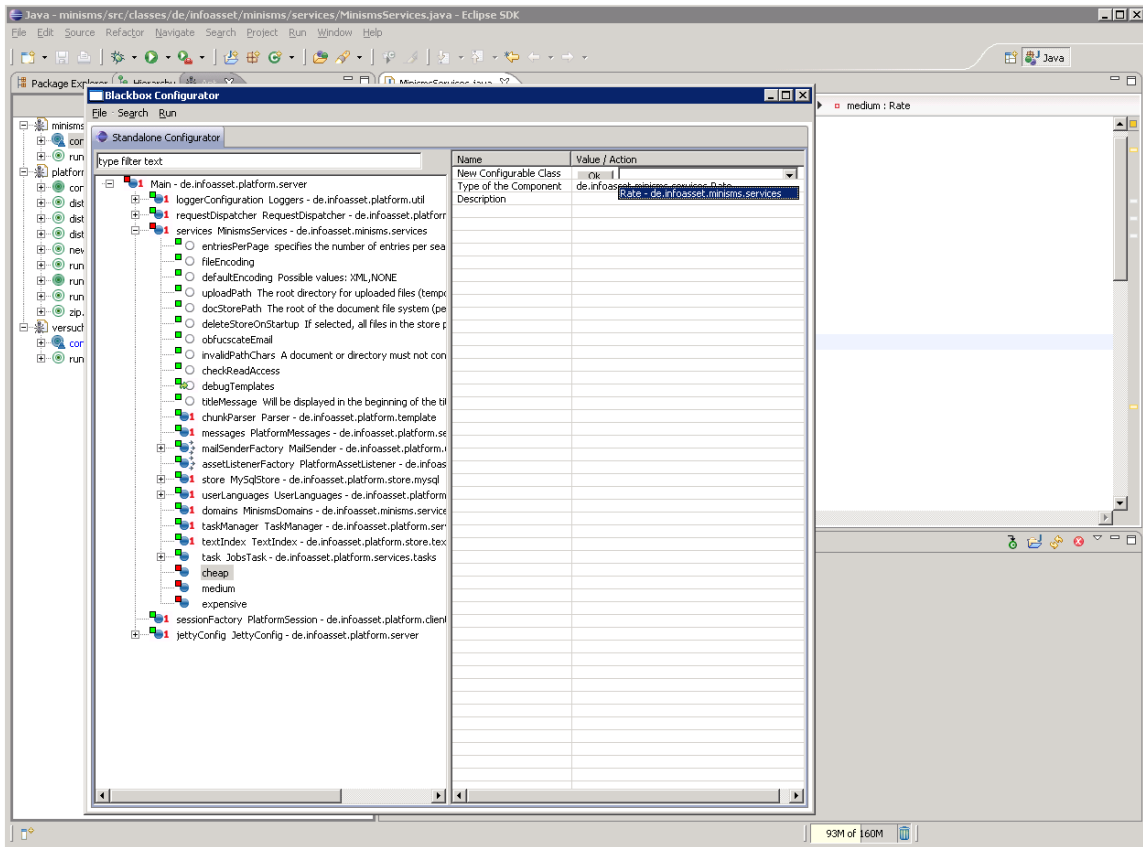


Figure 39: Configuring a Class

As you can see, the three **Rate** instances are decorated with a red square, which tells you that the configuration is incomplete in this respect: We still have to set the actual values into the configurable attributes. Go ahead and set them to something sensible. I set the (price, minimalVolume) pairs for 'cheap', 'medium' and 'expensive' to (50,10), (100,5) and (1000,0), respectively. After setting the values, save the configuration in the stand-alone Configurator by clicking on File->Save.

Excursion :

The configuration tree

When you start your Toro application, the framework will look for configurable class-instances in the application and put the values stored for them inside of **configuration.xml** into them. The way it goes about this is that it starts in the Main class of the 'platform-Toro' project (this is not part of your application, but of the framework), and looks for **@Association** annotations. Let us look at some of these **@Associations** as we find them in the Main class of 'platform-toro' in the `de.infoasset.platform.server` package.

```
public class Main extends AbstractInitializable implements PostInitializable { ...
```

```
    @Association
    private void setLoggerConfiguration(GenericLoggers object) {
    }

    @Association
    private void setRequestDispatcher(RequestDispatcher r) {
    }

    @Association
    private void setServices(GenericServices services) {
    }

    ...
}
```

These **@Associations** are different from those we have encountered so far in that they annotate methods, not configurable instances – what is up with that? Well, the methods decorated with the **@Association** all conform to the following pattern:

```
@Association
private void set[BranchName](ConfigurableClass dummy)
```

The framework will deal with **@Association** annotations connected to this kind of pattern in the following way: It will use the [BranchName] to set up a new branch with that name in the stand-alone Configurator tree, and it will look for other **@Association** or **@Property** annotations inside of the source files of the class-type gleaned from the parameter type passed in the function (**ConfigurableClass** in the above example) – all of this requires heavy use of reflection on the part of the framework, of course. The framework will actually do one more thing: In case **ConfigurableClass** is a Singleton, i.e. a class which is supposed to be instantiated only once, and instantiated by the framework, not the framework-user, the framework will now instantiate it. Finding out if a class is meant to be a Singleton is also done reflectively by a Toro convention, which says that Singletons should have an instance of the tell-tale attribute name *'instance'* of the type of the Singleton. This attribute shall of course reference the Singleton itself. If this is the case, the framework will instantiate the class and set *'instance'* to it.

If you would like to have your own Root-Branch in the Configurator, you would have to modify the **Main** class of the framework in 'platform-toro, with an annotated set-function, which is deprecated, because you should leave the framework untouched. Instead, hang your **Configurables** into the **Services** class of your own project.

3.20 Implementing Logic – extending Assets

We now have all the tools in hand to complete our application. From Figure 10) we can see that we have not yet implemented the creation of new **Partys**, viewing them, and deleting them. You can try writing a **NewHandler** for **Party** as an exercise, using the one for **Person** as a model. You should get something like this:

```
public class NewHandler extends Handler {

    final Line EDIT = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newParty.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
        }
    };

    private Party newParty;

    @Override
    public Station doBusinessLogic() throws Exception {
        newParty = Party.SCHEMA.createTransientAsset();
        return EDIT;
    }

}
```

There is absolutely nothing new here. Of course, it is important to place it in the correct package: `de.infoasset.minisms.assets.Party`.

As you can see, after creating our **Party** Asset, setting its id as a parameter, we go to the **EditHandler** (which we place in `de.infoasset.minisms.assets.Party`):

```
public class EditHandler extends Handler {

    private String id;

    private Party newParty;

    @Override
    public void getParameters(ParameterReader parameters) {
        id = parameters.getString("id");
    }

    final SimplePage EDITPAGE = new SimplePage() {
        @Override
        public Asset getAsset() {
            return newParty;
        }
    }

}
```

```

};

@Override
public Station doBusinessLogic() throws Exception {
    newParty = Party.SCHEMA.getAsset(id);
    return EDITPAGE;
}
}

```

This is just more boiler-plate code.

The corresponding HTML file `edit.htm` which must be placed in `\minisms\templates\standard\party\edit.htm` will however be slightly different from what we have seen before:

```

$header()$

<form method="post" action="/party/submit?id=$id.value()$"
class="Toro-edit-form">
    $editHeader()$
    $name.edit()$
    $location.edit()$
    $size.edit()$
    $begin.edit()$
    $editFooter()$
</form>

$footer()$

```

The diagram shows two callout boxes. The first box, labeled "Toro-Form Overhead", has a line pointing to the `$editHeader()$` substitution function in the HTML code. The second box, labeled ".edit()", has a line pointing to the `$name.edit()$`, `$location.edit()$`, `$size.edit()$`, and `$begin.edit()$` substitution functions.

.edit():

In contrast to the **Person** Edit-Form in which we only excluded a few properties which we didn't want to edit, we now have a form in which we explicitly *include* only a few properties which we wish the user to edit. So here, we use the `[$propertyname].edit()$` Substitution-Function, instead of the `$edit()$` Substitution-Function, which renders all properties. However, this comes at the price of having to explicitly include the Toro-Form Overhead:

Toro-Form Overhead:

In our previous Edit-Page, we used the `$edit()$` Substitution-Function, which takes care of including save&cancel buttons. We have to do that explicitly now, because the `[$propertyname].edit()$` substitution functions show up several times, and can therefore not be expected to display the buttons, as they would then also be displayed several times. Inserting the buttons is therefore done by the `$editHeader()$` and `$editFooter()$` Substitution-Functions (predefined in 'platform-toro').

Let us now write the **SubmitHandler** of **Party**, to which control is passed when submitting the above form. Per naming law, it must be placed in the `de.infoasset.minisms.handler.party` package:

```
public class SubmitHandler extends Handler {

    private String id;

    private Party newParty;

    Line VALID = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    Line INVALID = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newParty.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        newParty = Party.SCHEMA.getAsset(id);
        newParty.makeTransient();
        newParty.applyParameters();
        if (newParty.isValid()) {
            newParty.purchaseDate.set(new Date());
            newParty.organizer.set(GenericSessionLocal.getUser());
            newParty.rateCategory.set((newParty.organizer.getAsset().determineRate()));
            newParty.price.set(newParty.getRate().getPrice() * newParty.size.get());
            newParty.makePersistent();
            return VALID;
        } else {
            return INVALID;
        }
    }

    @Override
    public void getParameters(ParameterReader parameters) {
        id = parameters.getString("id");
    }
}

```

There are only two things worth pointing out:

Setting the Rate:

A newly created party is associated with a certain rate, which we save in the 'rateCategory' **DomainValueProperty** of the new Party. However, we must determine which rate applies – this depends on the total number of parties purchased so far by the customer. A function returning the correct **DomainValue**, **.determineRate()**, must be implemented in the **Person** Asset – we will do so right away. Let us assume it is in place already. We can then set the rate as shown, and continue with:

Calculating the Price:

The price is proportional to the size of the party and the price of the current rate. We retrieve 'size' of the **Party** with `newParty.size.get()`, but retrieving the rate is slightly more involved: Our configurable **Rates** correspond to **DomainValues**, which by themselves do not contain any information about the applicable prices: We will therefore need to define a function **.getRate()** which returns the correct **Rate** object in function of the **DomainValue** in the **Party** Asset. We then can call the **.getPrice()** method on that **Rate**.

So let us add the **.determineRate()** function to **Person**, and the **.getRate()** function to **Party**. Adding these methods should also remind you that you are in no way restricted in adding your own functionality to **BaseAssets**, and you will often do so:

So open the **Person** class from 'minisms' in the `de.infoasset.platform.assets` package and add the highlighted code:

```
public class Person extends BaseAsset {
    •
    •
    <snip>
    •
    •
    public DomainValue determineRate() {
        int totalParties = parties.count();
        if (totalParties > Services.INSTANCE().getCheap().getMinimalVolume()) {
            return RateDomain.cheap;
        } else if (totalParties > Services.INSTANCE().getMedium().getMinimalVolume()) {
            return RateDomain.medium;
        }
    }
}
```

```

        return RateDomain.expensive;
    }

    public static final AssetSchema<Person> SCHEMA = new AssetSche-
ma<Person>() {
        };

    public boolean isAdministrator() {
        return false;
    }
}

```

The only new Toro-thing here is the `.count()` method you can use on **Roles**: `parties.count()` will return the number of objects on the other side of the association. Here, we wish to find out how many parties the customer has purchased so far, and then check whether the number is high enough for one of the cheap rates. If it is, we return the corresponding **DomainValue**.

We now proceed to add the `.getRate()` function, which maps the rate **DomainValue** saved in 'rate' of **Party** to the corresponding (configurable) **Rate** object, which is what we actually need to determine the price. This is a functionality of **Party**, so open the **Party** Asset, and add the highlighted code:

```

public class Party extends BaseAsset {
    .
    .
    <snip>
    .
    .

    public Rate getRate() {
        if (this.rateCategory.get().equals(RateDomain.expensive))
            return Services.INSTANCE().getExpensive();
        else if (this.rateCategory.get().equals(RateDomain.medium))
            return Services.INSTANCE().getMedium();
        return Services.INSTANCE().getCheap();
    }

    public static final AssetSchema<Party> SCHEMA = new AssetSche-
ma<Party>();
}

```

3.21 Implementing Persistence – Deleting Assets

In chapter (3.18) you can see that in the list of the parties, the name is linked to a 'view' page, which we haven't implemented yet. It is here that we wish to see the details of the party, and be able to delete it.

Let us design the 'view' page of the **Party** asset first. It is called **view.htm** and must be placed in `minisms\templates\standard\party\view.htm`

```
$header() $  
  
<h1>Party "$name.value()"</h1>  
  
$name.show() $  
$location.show() $  
$size.show() $  
$begin.show() $  
  
<br>  
Total Price: $price$  
<br>  
[<a href="/party/delete?id=$id.value()">Delete Party</a>]  
[<a href="/party/list">View List</a>]  
<br>  
  
$footer() $
```

ViewHandler, placed in the `de.infoasset.minisms.handler.party` package, looks like this:

```
public class ViewHandler extends Handler {  
  
    private String id;  
  
    private Party party;  
  
    @Override  
    public void getParameters(ParameterReader parameters) {  
        id = parameters.getString("id");  
    }  
  
    final SimplePage HOME = new SimplePage() {  
        @Override  
        public void putSubstitutions(Template template) {  
            template.put("price", new PrintSubstitution() {  
                @Override  
                protected String print() {  
                    return Integer.toString((party.size.get()) * (party.getRate().getPrice()));  
                }  
            });  
        }  
    };  
  
    @Override  
    public Asset getAsset() {  
        return party;  
    }  
};  
  
@Override  
public Station doBusinessLogic() throws Exception {
```

```

        party = Party.SCHEMA.getAsset(id);
        return HOME;
    }
}

```

The only thing we don't know how to do yet – and it is very easy – is to delete an Asset. For this, we will implement the **DeleteHandler** which `view.htm` links to. It is a pageless Handler, which goes back to the party list after it is executed. Put it in the `de.infoasset.minisms.handler.party` package:

```

public class DeleteHandler extends Handler {

    String id;

    Party party;

    public void getParameters(ParameterReader parameters) {
        id = parameters.getString("id");
    }

    Line LIST = new Line() {
        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    public Station doBusinessLogic() throws Exception {
        party.remove();
        return LIST;
    }
}

```



3.22 Implementing Logic – Checking User Authorization

So far, we have not worried about security. However, security is supported by Toro. For example, a https connection is used if you call `useHttps()` on your **Forwarder**.

Another functionality of Toro consists in offering a `checkAccess()` method you can override in your **Handler**. In this method, you can make sure that the user trying to access the page has the proper authorization. For example, when deleting a **Party**, it should be impossible for a user who does not own the party to do so. If a malevolent user wanted to delete parties not created and owned by him, he could however do so by sending id's of parties not owned by him to the **DeleteHandler**. Relying on the malevolent user not knowing the id's of parties he doesn't own is not a very secure strategy. Let us implement a `checkAccess()` routine in **DeleteHandler** that throws an exception in case of such mischief:

```

public class DeleteHandler extends Handler {

```

```

String id;

Party party;

@Override
public void checkAccess() {
    party = Party.SCHEMA.getAssetNotNull(id);

    if(!GenericSessionLocal.getUser().equals(party.organizer.getAsset()))
    {
        throw new ProtectedActionException();
    }
}

public void getParameters(ParameterReader parameters) {
    id = parameters.getString("id");
}

Line LIST = new Line() {
    @Override
    public void next(Forwarder f) {
        f.go(ListHandler.class);
    }
};

public Station doBusinessLogic() throws Exception {
    party.remove();
    return LIST;
}
}

```

First, we retrieve the Asset from the Database through the **.getAssetNotNull()** method. This method throws an Exception of type **AssetNotFoundException** in case no Asset can be retrieved. This would only be the case if a malevolent user had changed the id parameter in the HTTP request. Toro will catch and deal with this kind of Exception, you do not have to provide any additional code.

In the next step, we check whether the current owner of the session is also the owner of the party. In case not, we throw a **ProtectedActionException**. Toro will also deal with this appropriately.

Try it: Create a party under one account, copy its id from the view page, then log in under another account, and point your browser to:

`http://localhost:8083/party/delete?id=<foreign party id>`

or

`http://localhost:8083/party/delete?id=<inexistent id>`

You will be redirected to a warning page stating the problem.

3.23 Implementing Logic – Logging Out

We have come full circle. The only thing that remains to be done is to log out. In chapter (3.15) we already placed an appropriate link in the `header.htm` web-page, and we now must implement the pageless Handler which will perform the logout. As is the case with the **LoginHandler**, the **LogoutHandler** does not belong to a specific Asset, and should therefore be placed in the top-level `de.infoasset.minisms.handler` package:

```
public class LogoutHandler extends Handler {
    Line LOGOUT = new Line() {
        public void next(Forwarder f) {
            f.go(LoginHandler.class);
        }
    };
    public Station doBusinessLogic() throws Exception {
        GenericSessionLocal.getSession().setUser(null);
        return LOGOUT;
    }
}
```

We return to the Login-Page - and to the end of this tutorial.

Appendix A: Listings

+Party.java

+:New Class/File

~:Modified Class/File

```
package de.infoasset.minisms.assets;

import de.infoasset.minisms.services.Domains;
import de.infoasset.minisms.services.Rate;
import de.infoasset.minisms.services.Services;
import de.infoasset.minisms.services.domains.RateDomain;
import de.infoasset.platform.assets.Person;
import de.infoasset.platform.services.asset.AssetSchema;

import de.infoasset.platform.services.asset.BaseAsset;
import de.infoasset.platform.services.asset.DomainValueProperty;
import de.infoasset.platform.services.asset.IntProperty;
import de.infoasset.platform.services.asset.OneRole;
import de.infoasset.platform.services.asset.Role;
import de.infoasset.platform.services.asset.StringProperty;
import de.infoasset.platform.services.asset.TimestampProperty;
import
de.infoasset.platform.services.asset.propertyValidators.NotNullValidat
or;
import
de.infoasset.platform.services.asset.propertyValidators.PropertyValida
tors;
import de.infoasset.platform.services.domains.Domain;
import de.infoasset.platform.services.domains.DomainValue;
import de.infoasset.platform.services.internationalization.Message;

public class Party extends BaseAsset {

    public final StringProperty name = new StringProperty() {

        @Override
        public Message getLabel() {
            return new Message() {

                String en = "Name";
            };
        }

        @Override
        protected void putValidators(PropertyValidators aggregator) {
            aggregator.add(new NotNullValidator());
        }

        @Override
        public int getMaxLength() {
            return 30;
        }
    };
};
```



```

public final StringProperty location = new StringProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Location";
        };
    }

    @Override
    protected void putValidators(PropertyValidators aggregator) {
        aggregator.add(new NotNullValidator());
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final IntProperty size = new IntProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Size";
        };
    }

    @Override
    protected void putValidators(PropertyValidators aggregator) {
        aggregator.add(new NotNullValidator());
    }
};

public final IntProperty price = new IntProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Price";
        };
    }

    @Override
    protected void putValidators(PropertyValidators aggregator) {
        aggregator.add(new NotNullValidator());
    }
};

public final TimestampProperty begin = new TimestampProperty() {
    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Begin";
        };
    }
};

```

```

        @Override
        protected void putValidators(PropertyValidators aggregator) {
            aggregator.add(new NotNullValidator());
        }
    };

    public final TimestampProperty purchaseDate = new TimestampProperty() {

        @Override
        public Message getLabel() {
            return new Message() {
                String en = "Purchase Date";
            };
        }
    };

    public final DomainValueProperty rateCategory = new DomainValueProperty() {

        @Override
        public Message getLabel() {
            return new Message() {
                String en = "Rate";
            };
        }

        @Override
        public DomainValue getDefaultDomainValue() {
            return RateDomain.expensive;
        }

        @Override
        public Domain getDomain() {
            return Domains.RateDomain;
        }
    };

    /* Party * <-> 1 Person */
    final public OneRole<Person> organizer = new OneRole<Person>() {
        @Override
        public Role otherRole() {
            return Person.SCHEMA.prototype().parties;
        }
    };

    public Rate getRate() {
        if (this.rateCategory.get().equals(RateDomain.expensive))
            return Services.INSTANCE().getExpensive();
        else if (this.rateCategory.get().equals(RateDomain.medium))
            return Services.INSTANCE().getMedium();
        return Services.INSTANCE().getCheap();
    }

    public static final AssetSchema<Party> SCHEMA = new AssetSchema<Party>();
}

```

+RateDomain.java

```
package de.infoasset.minisms.services.domains;

import de.infoasset.platform.services.domains.Domain;
import de.infoasset.platform.services.domains.DomainValue;
import de.infoasset.platform.services.internationalization.Message;

public class RateDomain extends Domain {

    public static final DomainValue expensive = new DomainValue() {

        @Override
        protected Message nameMessage() {
            return new Message() {
                String en = "expensive";
            };
        }
    };

    public static final DomainValue medium = new DomainValue() {

        @Override
        protected Message nameMessage() {
            return new Message() {
                String en = "medium";
            };
        }
    };

    public static final DomainValue cheap = new DomainValue() {

        @Override
        protected Message nameMessage() {
            return new Message() {
                String en = "cheap";
            };
        }
    };
}
```

~Domains.java

```
package de.infoasset.minisms.services;

import de.infoasset.minisms.services.domains.RateDomain;
import de.infoasset.minisms.services.domains.Title;
import de.infoasset.platform.services.domains.Domain;
import de.infoasset.platform.services.domains.GenericDomains;

public class Domains extends GenericDomains {

    public static final Domain Title = new Title();

    public static final Domain RateDomain = new RateDomain();

}
```

+Person.java

```
package de.infoasset.platform.assets;

import de.infoasset.minisms.assets.Party;
import de.infoasset.minisms.services.Messages;
import de.infoasset.minisms.services.Services;
import de.infoasset.minisms.services.domains.RateDomain;
import de.infoasset.platform.assets.Person;
import de.infoasset.platform.services.asset.BaseAsset;
import de.infoasset.platform.services.asset.AssetSchema;
import de.infoasset.platform.services.asset.BooleanProperty;
import de.infoasset.platform.services.asset.ManyRole;
import de.infoasset.platform.services.asset.PasswordProperty;
import de.infoasset.platform.services.asset.Role;
import de.infoasset.platform.services.asset.StringProperty;
import de.infoasset.platform.services.asset.TimestampProperty;
import
de.infoasset.platform.services.asset.propertyValidators.MinimalLengthV
alidator;
import
de.infoasset.platform.services.asset.propertyValidators.NotNullValidat
or;
import
de.infoasset.platform.services.asset.propertyValidators.PropertyValida
tors;
import
de.infoasset.platform.services.asset.propertyValidators.RegExpValidato
r;
import de.infoasset.platform.services.domains.DomainValue;
import de.infoasset.platform.services.internationalization.Message;

public class Person extends BaseAsset {

    public final BooleanProperty isAdministrator = new BooleanProper-
ty() {

        @Override
        public boolean getDefaultValue() {
            return false;
        }

        @Override
        public boolean hasDefaultValue() {
            return true;
        }

        @Override
        public Message getLabel() {
            return new Message() {

                String en = "Is this Person an Administrator?";
            };
        }
    };

    public final StringProperty eMail = new StringProperty() {

        @Override
```

```

public Message getLabel() {
    return new Message() {

        String en = "E-Mail Address";
    };
}

@Override
public int getMaxLength() {

    return 100;
}

@Override
protected void putValidators(PropertyValidators aggregator) {
    aggregator.add(new NotNullValidator() {
    });
    aggregator.add(new RegExpValidator() {
        @Override
        protected Message getErrorMessage() {

            return Messages.validator_invalid_email;
        }

        @Override
        public String getRegExp() {
            return "^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[a-z0-9-
]+(\\.[a-z0-9-]+)*$";
        }
    });
}

};

public final StringProperty firstName = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "First Name";
        };
    }

    @Override
    public boolean isFulltextIndexed() {
        return true;
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final StringProperty lastName = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

```

```

        String en = "Last Name";
    };
}

@Override
protected void putValidators(PropertyValidators aggregator) {
    aggregator.add(new NotNullValidator());
    aggregator.add(new MinimalLengthValidator(3));
}

@Override
public boolean isFulltextIndexed() {
    return true;
}

@Override
public int getMaxLength() {
    return 30;
}
};

public final StringProperty street = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "Street";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final StringProperty zipCode = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {

            String en = "ZipCode";
        };
    }

    @Override
    public int getMaxLength() {
        return 10;
    }
};

public final StringProperty city = new StringProperty() {

    @Override
    public Message getLabel() {

```

```

        return new Message() {
            String en = "City";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final StringProperty country = new StringProperty() {

    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Country";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final StringProperty password = new PasswordProperty() {

    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Password";
        };
    }

    @Override
    public int getMaxLength() {
        return 30;
    }
};

public final TimestampProperty membershipDate = new TimestampProperty() {

    @Override
    public Message getLabel() {
        return new Message() {
            String en = "Date of Membership";
        };
    }
};

```



```

/* Party * <-> 1 Person */
final public ManyRole<Party> parties = new ManyRole<Party>() {
    @Override
    public Role otherRole() {
        return Party.SCHEMA.prototype().organizer;
    }
};

public DomainValue determineRate() {
    int totalParties = parties.count();
    if (totalParties > Services.INSTANCE().getCheap().getMinimalVolume()) {
        return RateDomain.cheap;
    } else if (totalParties > Services.INSTANCE().getMedium().getMinimalVolume()) {
        return RateDomain.medium;
    }
    return RateDomain.expensive;
}

public static final AssetSchema<Person> SCHEMA = new AssetSchema<Person>() {
};

public boolean isAdministrator() {
    return false;
}
}

```

~Services.java

```
package de.infoasset.minisms.services;

import java.util.Date;
import de.infoasset.imf.blackbox.Association;
import de.infoasset.minisms.assets.Party;
import de.infoasset.minisms.services.domains.RateDomain;
import de.infoasset.platform.assets.Person;
import de.infoasset.platform.services.GenericServices;
import de.infoasset.platform.services.asset.GenericAssetListener;

public class Services extends GenericServices {
    private static Services instance;

    public static Services INSTANCE() {
        return instance;
    }

    @Association
    private Rate cheap;

    @Association
    private Rate medium;

    @Association
    private Rate expensive;

    public Rate getCheap() {
        return cheap;
    }

    public Rate getMedium() {
        return medium;
    }

    public Rate getExpensive() {
        return expensive;
    }

    @Override
    public void initData() {
        Person tester = Person.SCHEMA.createAsset();

        tester.eMail.set("mymail@myserver.com");
        tester.firstName.set("Sebastian");
        tester.lastName.set("Henckel");
        tester.street.set("Kaiserdamm 28");
        tester.zipCode.set("14057");
        tester.city.set("Berlin");
        tester.country.set("Germany");
        tester.password.set("prettyplease");
        tester.membershipDate.set(new Date(108, 0, 1));
        tester.isAdministrator.set(false);

        Party fiesta = Party.SCHEMA.createAsset();

        fiesta.name.set("Test Party");
    }
}
```

```

        fiesta.location.set("Irish pub");
        fiesta.size.set(100);
        fiesta.begin.set(new Date(108, 0, 1));
        fiesta.rateCategory.set(RateDomain.cheap);
        fiesta.purchaseDate.set(new Date());

        tester.parties.create(fiesta);

        GenericAssetListener.commit();
    }

    @Override
    protected void initSchemas() {
        initSchema(Person.class);
        initSchema(Party.class);
    }

    @Override
    protected void initMixinBaseSchemas() {
    }

    @Override
    public String getWelcome() {
        return "/login";
    }
}

```

+minims\templates\standard\functions\header.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; char-
set=UTF-8"/>
    <script type="text/javascript" src="/__/jquery.js"></script>
    <script type="text/javascript" src="/__/platform.js"></script>
    <link rel="stylesheet" type="text/css" href="/__/toro-
platform.css"/>
    <title>Toro Tutorial</title>
  </head>
  <body>
    ${anonymous()}$
    <h1>Toro Tutorial</h1>
    ${anonymous[$
    <p>
      You are logged in as <a
href="/person/edit?id=${currentUserId()}$">${currentUserName()}$</a> [<a
href="/logout">Logout</a>]
    </p>
    <h1>Toro Tutorial</h1>
    $anonymous]$
```

`+\\minisms\\templates\\standard\\functions\\footer.htm`

`</body>`

+Functions.java

```
package de.infoasset.minisms.handler;

import de.infoasset.platform.assets.Person;
import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.template.FunctionParameters;
import de.infoasset.platform.template.PrintSubstitution;
import de.infoasset.platform.template.TemplateFinder;
import de.infoasset.platform.template.TemplateSubstitution;

public class Functions {
    public static TemplateSubstitution header(FunctionParameters pa-
rams) {
        return new TemplateSubstitution() {

            @Override
            public void specifyTemplate(TemplateFinder templateName) {
                templateName.useStaticName("functions/header");
            }
        };
    }

    public static TemplateSubstitution footer(FunctionParameters pa-
rams) {
        return new TemplateSubstitution() {
            @Override
            public void specifyTemplate(TemplateFinder templateName) {
                templateName.useStaticName("functions/footer");
            }
        };
    }

    public static PrintSubstitution currentUserName(FunctionParameters
params) {
        return new PrintSubstitution() {

            @Override
            protected String print() {
                Person p = GenericSessionLocal.getUser();
                return p == null ? "" : p.eMail.get();
            }
        };
    }
}
```

+minisms\templates\standard\login.htm

\$header()\$

You must login....

\$errorMessage\$

```
<p>
  <form action="/submitLogin">
    Login: <input type="text" name="login"/>Password: <input
type="password" name="password"/>
    <input type="submit" value="Submit"/>
  </form>
  [<a href="/person/new">New Account</a>]
</p>
```

\$footer()\$

+LoginHandler.java

```
package de.infoasset.minisms.handler;

import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.SimplePage;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.template.PrintSubstitution;
import de.infoasset.platform.template.Template;

public class LoginHandler extends Handler {

    private String errorMessage;

    @Override
    public void getParameters(ParameterReader parameters) {
        errorMessage = parameters.getString("errorMessage");
    }

    final SimplePage LOGINPAGE = new SimplePage() {

        @Override
        public void putSubstitutions(Template template) {
            template.put("errorMessage", new PrintSubstitution() {
                @Override
                public String print() {
                    return Handler.notNull(errorMessage);
                }
            });
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        System.out.println("LoginHandler dBL()");
        return LOGINPAGE;
    }
}
```


+SubmitLoginHandler.java

```
package de.infoasset.minisms.handler;

import de.infoasset.minisms.handler.party.ListHandler;
import de.infoasset.platform.assets.Person;
import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.client.ParameterWriter;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.services.internationalization.Message;
import de.infoasset.platform.store.QueryEquals;

public class SubmitLoginHandler extends Handler {

    private String login;

    private String password;

    @Override
    public void getParameters(ParameterReader parameters) {
        login = parameters.getString("login");
        password = parameters.getString("password");
    }

    final Line SUCCESS = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    final Line FAILURE = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(LoginHandler.class);
        }

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setMessage("errorMessage", noLogin, new Message() {

                String en = "Login missing.";
            });
            parameters.setMessage("errorMessage", notRegistered, new Message() {

                String en = "You entered an unknown e-mail address. Please try another e-mail address.";
            });
            parameters.setMessage("errorMessage", pwdError, new Message() {
```

```

        String en = "You entered a wrong password. Please try
again.";
    });
}
};

private Person candidate;

private boolean noLogin;

private boolean notRegistered;

private boolean pwdError;

@Override
public Station doBusinessLogic() throws Exception {
    System.out.println("SubmitLoginHandler dBL()");
    if (login == null || login.length() == 0) {
        noLogin = true;
        return FAILURE;
    }
    candidate = Person.SCHEMA.findSingleAsset(new QueryE-
quals(Person.SCHEMA.prototype().eMail, login));
    if (candidate == null) {
        notRegistered = true;
        return FAILURE;
    }
    if (!candidate.password.get().equals(password)) {
        pwdError = true;
        return FAILURE;
    }
    GenericSessionLocal.getSession().setUser(candidate);
    return SUCCESS;
}
}
}

```

`+/minisms/templates/standard/person/edit.htm`

```
$header() $
```

```
<form method="post" action="/person/submit?id=$id.value()" $"  
class="toro-edit-form">  
  $edit(exclude=isAdministrator, membershipDate) $  
</form>
```

```
$footer() $
```

+EditHandler (Person)

```
package de.infoasset.minisms.handler.person;

import de.infoasset.platform.assets.Person;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.SimplePage;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.services.asset.Asset;

public class EditHandler extends Handler {

    private String accountId;

    private Person account;

    @Override
    public void getParameters(ParameterReader parameters) {
        accountId = parameters.getString("id");
    }

    final SimplePage EDITPAGE = new SimplePage() {
        @Override
        public Asset getAsset() {
            return account;
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        System.out.println("EditNewAccountHandler dBL()");
        account = Person.SCHEMA.getAsset(accountId);
        return EDITPAGE;
    }
}
```

+SubmitHandler (Person)

```
package de.infoasset.minisms.handler.person;

import de.infoasset.minisms.handler.party.ListHandler;
import de.infoasset.platform.assets.Person;
import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.client.ParameterWriter;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;

public class SubmitHandler extends Handler {

    private String accountId;

    private Person newAccount;

    @Override
    public void getParameters(ParameterReader parameters) {
        accountId = parameters.getString("id");
    }

    final Line VALID = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    Line INVALID = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newAccount.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        System.out.println("SubmitPersonHandler dBL()");
        newAccount = Person.SCHEMA.getAsset(accountId);
        newAccount.makeTransient();
        newAccount.applyParameters();
        if (newAccount.isValid()) {
            newAccount.makePersistent();
            GenericSessionLocal.getSession().setUser(newAccount);
            return VALID;
        } else {
            return INVALID;
        }
    }
}
```

+NewHandler (Person)

```
package de.infoasset.minisms.handler.person;

import de.infoasset.platform.assets.Person;
import de.infoasset.platform.client.ParameterWriter;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;

public class NewHandler extends Handler {

    final Line EDIT = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newAccount.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
            f.noRedirect();
        }
    };

    private Person newAccount;

    @Override
    public Station doBusinessLogic() throws Exception {
        System.out.println("CreateNewAccount dBL()");
        newAccount = Person.SCHEMA.createTransientAsset();
        return EDIT;
    }
}
```

+ \minims\templates\standard\party\list.htm

```
$header() $  
  
<h1>"Parties" Page of $user$</h1>  
<table>  
  <tr>  
    <th>  
      Title  
    </th>  
    <th>  
      Location  
    </th>  
    <th>  
      Begin  
    </th>  
    <th>  
      Price  
    </th>  
    <th>  
      Purchase Date  
    </th>  
  </tr>  
  $[parties p$  
  <tr>  
    <td>  
      <a  
href="/party/view?id=$p.id.value() $">$p.name.value() $</a>  
    </td>  
    <td>  
      $p.location.value() $  
    </td>  
    <td>  
      $p.begin.value() $  
    </td>  
    <td>  
      $p.price.value() $  
    </td>  
    <td>  
      $p.purchaseDate.value() $  
    </td>  
  </tr>  
  $parties]$  
</table>  
  
[<a href="/party/new">Create a new Party</a>]  
  
$footer() $
```

+ListHandler.java (Party)

```
package de.infoasset.minisms.handler.party;

import java.util.Iterator;
import de.infoasset.minisms.assets.Party;
import de.infoasset.platform.assets.Person;
import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.SimplePage;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.services.asset.Asset;
import de.infoasset.platform.store.QueryEquals;
import de.infoasset.platform.store.SortingCriterion;
import de.infoasset.platform.template.ListSubstitution;
import de.infoasset.platform.template.PrintSubstitution;
import de.infoasset.platform.template.Template;

public class ListHandler extends Handler {
    private Person user = GenericSessionLocal.getSession().getUser();

    final SimplePage LISTPAGE = new SimplePage() {

        @Override
        public void putSubstitutions(Template template) {
            template.put("parties", new ListSubstitution() {

                Party currentParty;

                Iterator<Party> myParties;

                @Override
                public void start() {
                    QueryEquals q = new QueryE-
quals(Party.SCHEMA.prototype().organizer.getAttributeSignature(), user.id.get());
                    q.addSortingCriterion(new SortingCrite-
rion(Party.SCHEMA.prototype().begin, SortingCriterion.ASCENDING));
                    myParties = Party.SCHEMA.queryAssets(q);
                }

                @Override
                public void next() {
                    currentParty = myParties.next();
                }

                @Override
                public Asset getCurrentAsset() {
                    return currentParty;
                }

                @Override
                public boolean hasNext() {
                    return myParties.hasNext();
                }

            });
            template.put("user", new PrintSubstitution() {
                @Override
```



```
        protected String print() {
            return user.eMail.get();
        }
    });
}
};

@Override
public Station doBusinessLogic() throws Exception {
    return LISTPAGE;
}
}
```

+Rate.java

```
package de.infoasset.minisms.services;

import de.infoasset.imf.blackbox.Configurable;
import de.infoasset.imf.blackbox.Property;

public class Rate implements Configurable {

    @Property
    private int price;

    @Property
    private int minimalVolume;

    public int getMinimalVolume() {
        return minimalVolume;
    }

    public int getPrice() {
        return price;
    }
}
```

+NewHandler.java (Party)

```
package de.infoasset.minisms.handler.party;

import de.infoasset.minisms.assets.Party;
import de.infoasset.platform.client.ParameterWriter;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;

public class NewHandler extends Handler {

    final Line EDIT = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newParty.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
        }
    };

    private Party newParty;

    @Override
    public Station doBusinessLogic() throws Exception {
        newParty = Party.SCHEMA.createTransientAsset();
        return EDIT;
    }
}
```

`+/minisms/templates/standard/party/edit.htm`

```
$header() $  
  
<form method="post" action="/party/submit?id=$id.value()" $"  
class="toro-edit-form">  
  $editHeader() $  
  
  $name.edit() $  
  $location.edit() $  
  $size.edit() $  
  $begin.edit() $  
  
  $editFooter() $  
</form>  
  
$footer() $
```

+EditHandler.java (Party)

```
package de.infoasset.minisms.handler.party;

import de.infoasset.minisms.assets.Party;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.SimplePage;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.services.asset.Asset;

public class EditHandler extends Handler {

    private String id;

    private Party newParty;

    @Override
    public void getParameters(ParameterReader parameters) {
        id = parameters.getString("id");
    }

    final SimplePage EDITPAGE = new SimplePage() {
        @Override
        public Asset getAsset() {
            return newParty;
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        newParty = Party.SCHEMA.getAsset(id);
        return EDITPAGE;
    }
}
```

+SubmitHandler.java (Party)

```
import java.util.Date;

import de.infoasset.minisms.assets.Party;
import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.client.ParameterWriter;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;

public class SubmitHandler extends Handler {

    private String id;

    private Party newParty;

    Line VALID = new Line() {

        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    Line INVALID = new Line() {

        @Override
        public void setParameters(ParameterWriter parameters) {
            parameters.setString("id", newParty.id.get());
        }

        @Override
        public void next(Forwarder f) {
            f.go(EditHandler.class);
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        newParty = Party.SCHEMA.getAsset(id);
        newParty.makeTransient();
        newParty.applyParameters();
        if (newParty.isValid()) {
            newParty.purchaseDate.set(new Date());
            newParty.organizer.set(GenericSessionLocal.getUser());
            newParty.rateCategory.set((newParty.organizer.getAsset().determineRate()));
            newParty.price.set(newParty.getRate().getPrice() * newParty.size.get());
            newParty.makePersistent();
            return VALID;
        } else {
            return INVALID;
        }
    }
}
```

```
@Override
public void getParameters(ParameterReader parameters) {
    id = parameters.getString("id");
}
}
```

`+/minisms/templates/standard/party/view.htm`

```
$header() $  
  
<h1>Party "$name.value()"</h1>  
  
$name.show() $  
$location.show() $  
$size.show() $  
$begin.show() $  
  
<br>  
Total Price: $price$  
<br>  
[<a href="/party/delete?id=$id.value()">Delete Party</a>]  
[<a href="/party/list">View List</a>]  
<br>  
  
$footer() $
```


+ViewHandler.java (Party)

```
package de.infoasset.minisms.handler.party;

import de.infoasset.minisms.assets.Party;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.SimplePage;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.services.asset.Asset;
import de.infoasset.platform.template.PrintSubstitution;
import de.infoasset.platform.template.Template;

public class ViewHandler extends Handler {

    private String id;

    private Party party;

    @Override
    public void getParameters(ParameterReader parameters) {
        id = parameters.getString("id");
    }

    final SimplePage PARTYPAGE = new SimplePage() {
        @Override
        public void putSubstitutions(Template template) {
            template.put("price", new PrintSubstitution() {
                @Override
                protected String print() {
                    return Integer.toString((party.size.get()) * (party.getRate().getPrice()));
                }
            });
        }

        @Override
        public Asset getAsset() {
            return party;
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        party = Party.SCHEMA.getAsset(id);
        return PARTYPAGE;
    }
}
```

+DeleteHandler.java (Party)

```
package de.infoasset.minisms.handler.party;

import de.infoasset.minisms.assets.Party;
import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.client.ParameterReader;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;
import de.infoasset.platform.services.asset.ProtectedActionException;

public class DeleteHandler extends Handler {

    String id;

    Party party;

    @Override
    public void checkAccess() {
        party = Party.SCHEMA.getAssetNotNull(id);
        if (!GenericSessionLocal.al.getUser().equals(party.organizer.getAsset())) {
            throw new ProtectedActionException();
        }
    }

    @Override
    public void getParameters(ParameterReader parameters) {
        id = parameters.getString("id");
    }

    Line LIST = new Line() {
        @Override
        public void next(Forwarder f) {
            f.go(ListHandler.class);
        }
    };

    @Override
    public Station doBusinessLogic() throws Exception {
        party.remove();
        return LIST;
    }
}
```

+LogoutHandler.java

```
package de.infoasset.minisms.handler;

import de.infoasset.platform.client.GenericSessionLocal;
import de.infoasset.platform.handler.Forwarder;
import de.infoasset.platform.handler.Handler;
import de.infoasset.platform.handler.Line;
import de.infoasset.platform.handler.Station;

public class LogoutHandler extends Handler {

    Line LOGOUT = new Line() {

        public void next(Forwarder f) {
            f.go(LoginHandler.class);
        }
    };

    public Station doBusinessLogic() throws Exception {
        GenericSessionLocal.getSession().setUser(null);
        return LOGOUT;
    }
}
```

References

- [AE01] Alfons Kemper and André Eickler, *Datenbanksysteme*, Oldenbourg, 2001
- [Bü07] Thomas Büchner, *Introspektive Modellgetriebene Softwareentwicklung*, Dissertation Technische Universität München, 2007
- [Bv08] Berliner Verkehrs Gesellschaft,
<http://www.bvg.de/index.php/de/Bvg/Detail/folder/195/id/2163/nb/1/name/BVG+fahinfo+SMS>, Accessed: 12. May 2008
- [CI08a] Clickatell Ltd, http://www.clickatell.com/downloads/http/Clickatell_HTTP.pdf,
Accessed: 30. May 2008
- [CI08b] Clickatell Ltd,
http://www.clickatell.com/downloads/Clickatell_SA_shortcode_MO_technical_guide.pdf,
, Accessed: 2. June 2008
- [DV07] Dialog Consult / VATM, Der deutsche Telekommunikationsmarkt –Zehn Jahre Liberalisierung im Festnetzmarkt, <http://www.vatm.de/content/studien/inhalt/16-10-2007.pdf>, Accessed: 12. May 2008
- [Ec06] Bruce Eckel, *Thinking in Java*, 4th Edition, Prentice Hall, 2006
- [Op08] OpenIT GmbH, <http://www.openit.de/preisliste-sms.html>, Accessed: 12. May 2008
- [Go08] Jan Goyvaerts, <http://www.regular-expressions.info/quickstart.html>, Accessed 21. July 2008
- [Go08a] Google Inc.,
<http://www.google.com/support/calendar/bin/topic.py?topic=13747&intention=13730>,
Accessed: 12. May 2008
- [Go08b] Google Inc., http://www.google.com/intl/en_us/mobile/default/sms/index.html,
Accessed: 27. August 2008
- [Le08] Leo GmbH, http://dict.leo.org/pages.ende/sms_de.html?p=ende&lang=de, Accessed: 13. May 2008
- [Se08] SelfHTML e.V., <http://www.selfhtml.org>, Accessed 30. August 2008