



From Technologies to Solutions

Apache OFBiz Development

The Beginner's Tutorial

Using Services, Entities, and Widgets to build custom ERP
and CRM systems

Jonathon Wong

Rupert Howell

[PACKT]
PUBLISHING

Table of Content

Preface	1
Chapter 1: Getting Started with OFBiz	9
Getting the OFBiz Code	9
Downloading and Installing SVN	10
Downloading TortoiseSVN	10
Using SVN to Get OFBiz	11
Our OFBiz Workspace—First Look	13
Installing the Java Development Kit (JDK)	14
Downloading JDK 5.0	14
Installing JDK 5.0	15
Downloading OFBiz Ready to Launch	16
Setting Up an Eclipse Project	16
Using Derby—the OFBiz Stock Database	19
Compiling OFBiz and Loading the Data	20
Verifying the Installation Process	20
The Compilation Portion	20
The Data Loading Portion	21
Some Pre-Ignition Processes we won't do Twice	21
Backing Up the Derby Data Files	22
Running OFBiz	22
Allocating Memory for OFBiz	22
Starting OFBiz and Verifying the Start-Up Process	24
Seeing the Logs in Real Time	25
Possible Problems with Start-Up	25
Switching Off OFBiz	25
Our First Tour of OFBiz	26
Accessing OFBiz	26
Exploring the Webapp "ecommerce"	28
Let's Buy Something	28
Exploring the Webapp "order"	32

Table of Contents

Receiving Payment	32
Fulfilling the Order	34
Invoice Automatically Generated with Payment Applied	34
End-to-End Tour Completed	34
Summary	35
Chapter 2: Working with OFBiz	37
Adding Our First Field	37
Changing the Data	39
Editing the Entity Definition	39
Updating the Database	40
Changing the Looks	40
Editing the User-Interface	41
Checking Our Changes	41
Changing the Flow	42
Rewiring the "Save" (Update Postal Address) Button	42
Creating the New Widget Screen	43
Creating the FTL File	44
More to the Flow	45
Some Changes Possible with Engines Running	47
Resetting Our Play Area Quickly	48
Skipping Some Pre-Ignition Processes	49
Restoring Derby Data Files	49
Removing the Web Server (Catalina) Work Files	49
Updating the Database with Our Data Entity Changes	49
Showing Off Our Spanking New OFBiz Installation	50
Tripping Up Our Plan	50
Storing a Save-Point to Dramatically Ease Testing	50
Archiving Derby Data Files and Web Server Work Files	51
Restoring a Save-Point	52
Restoring the Derby Data Files	52
Restoring the Web Server Work Files	52
Computer, Run Scenario A from Last Save-Point	52
The Structure of OFBiz in General	53
Components in OFBiz	53
Referencing Components in OFBiz	54
Creating Our Own OFBiz Component	55
Creating the Component	55
Using Our Component	55
Cleaning Up Our Mess in the "party" Component	56
Converting the BeanShell to a Java Event	57
Clean Extension Strategies Employed	59
Checking that Our Move was Successful	60
A Bit More Mess Remains	60
Webapps in OFBiz	60
Creating Our First Webapp	61

Table of Contents

Webapp URIs in this Book	62
Testing Our First Webapp	63
The Model-View-Controller Architectural Pattern	63
The MVC in Plain English	63
The Model in OFBiz	64
The View in OFBiz	64
The Controller in OFBiz	65
Other Files in an OFBiz Component	66
Summary	67
Chapter 3: Screen Widgets	69
Equipping Our Webapp with a Screen Widget View Handler	69
Using the Screen Widget View Handler	71
Files and Locations	71
Creating Our First Screen Widget	71
Defining a Screen Widget	71
Informing the Control Servlet about the Screen Widget	72
Referencing Screen Widgets	73
Uniform Pattern of Flow in OFBiz	73
Seeing Our First Screen Widget	74
The Anatomy of the <section> Element	74
Our First Conditional Screen Widget	74
Element Order in the controller.xml	75
Inform the Control Servlet about the Screen Widget	75
If-Then-Else Structure	75
The <if-condition> Element	76
The Then—<actions> and <widgets> Elements	76
The Else—<fail-widgets> Element	78
The Minimum <section>	78
Sending Parameters with Requests	79
Seeing Our Conditional Screen Widget	79
Screen Widget Context and Variables	79
Utility Objects in Context	80
Nested Sections for Nested Conditions	85
Organizing a Large Screen into Smaller Screens	88
The Global Context Revisited	90
Outer Contexts Visible to Nested Ones	92
Screen Widget's Integration with FreeMarker	93
Cleaning Up in the "party" Component	94
Commenting Changes to the Core Code	96
Screen Widgets as Templates	97
A Candidate for Templating	97
Creating the Header	98
Creating the Footer	98

Table of Contents

Using Our Header and Footer	99
Seeing Our First Well-Formed XHTML Document	99
Using Decorator Screen Widgets for Templating	100
Creating a XHTML Decorator Screen	100
Using the XHTML Decorator Screen	100
Seeing Our First Decorator in Action	101
Multiple Content Slots	102
Creating the First Half of the Header	102
Creating the Second Half of the Header	102
Adding a Content Slot to a Decorator	103
Using Our Multi-Slot Decorator	103
Seeing Our First Multi-Slot Decorator	104
Nesting Decorators	104
Top-Down Approach (delegation)	104
The Bottom-Up Approach (Vertical Stack)	106
Using Both Approaches	107
Summary	109
Chapter 4: Form Widgets	111
Files and Locations	112
Creating Our First Form Widget	112
Creating the Containing Screen Widget	112
Referencing Form Widgets	113
Create the Form Widget	113
Seeing Our First Form	114
Understanding the Form Attributes	114
Minimum Requirements to Use Form Widgets	115
Including the Minimal Requirements	116
Form Processing via Request Event	116
Java Events	117
Submitting and Processing Our First Form	118
The "list" Type Form Widget	119
Creating the Containing Screen	119
Adding Form Processing Code	120
Publishing the Form	121
Seeing Our First "list" Type Form	122
The "multi" Type Form Widget	122
Creating the Containing Screen	122
Loading Data for the Form	123
Publishing the Form	124
Creating the Form-Processing Logic	125
Seeing Our First "multi" Type Form	125
Alternative Targets in Two-Target Forms	126
Creating the Form	126

Table of Contents

Creating the Containing Screen	127
Publishing the Two-Target Form	127
Seeing Our First Two-Target Form	128
Row-Level Actions	128
Creating the Form	129
Creating the Containing Screen	129
Publishing the Form	130
Seeing the Form in Action	130
Summary	131
Chapter 5: Other View Element Types in Screen Widgets	133
Menu Widgets	134
Creating Our First Menu Widget	134
Including Our Menu Widget in Our Screen Widgets	135
Understanding the Menu Item Attributes	135
Including the Menu Widget via a Decorator Screen Widget	135
Add Screen Widget "ConditionalScreen" to the Menu	136
Sub-Menus and Conditional-Menu Items	137
Pre-processing Actions for Menu Widgets	139
FreeMarker	142
As Decorator Templates	142
Displaying Dynamically Created List Variables	143
Iterating Through the List	144
Bringing it All Together	145
The User-Interface Labels	146
Adding the appheader	148
Summary	150
Chapter 6: The Controller	151
How OFBiz Hears Our Requests—The Control Servlet	151
Defining a Control Servlet for a Webapp	152
Using the Control Servlet	153
Funnelling All Requests to the Single Control Servlet	153
Defining Needed Utility Objects for the Control Servlet	154
GenericDelegator Object	154
The GenericDispatcher Object	155
Some Background on Servlets	156
Programming a Control Servlet	156
Logging into Our Learning Application	156
Specifying Handlers	159
Request Maps	160
Knocking on the Right Doors	160
Security—Before Answering the Door	160
The https Attribute	160

Table of Contents

The auth Attribute	162
The direct-request Attribute	163
Event—Determining a Response	165
Java Events	165
Response—Defining Various Responses	166
View Maps	172
Summary	173
Chapter 7: Entities, View Entities, and Extended Entities	175
Entities	176
The Structure of the Data Model	176
Referencing Fields of an Entity	176
OFBiz Uses Relational Database Management Systems	176
Curious Trivia about the Relational Model	177
Entity Engine Concepts	177
Datasources	177
Entity Delegators	178
Entity Groups	179
Defining Our First Entity	179
Assigning Our Entity to an Entity Group	180
Loading Our Entity into the Entity Engine	180
Seeing Our First Entity	181
Using Our First Entity	183
Creating a Drop-Down	184
Populating the Drop-Down	186
Expiring a Value	187
Un-Expiring a Value	187
Anatomy of an <entity> Element	187
The <field> Element	188
The <prim-key> Element	189
The <relation> Element	189
The <index> Element	194
Relation Types	196
One-to-One Relations	196
One-to-Many Relations	196
One-to-One Relations with No Foreign Keys	199
View Entities	200
Anatomy of a <view-entity>	201
The <member-entity> Element	201
The <alias> and <alias-all> Elements	202
The <view-link> Element	206
The <relation> Element	209
Applying Functions on Fields	209
Counting the Number of Records	209
Counting Distinct Records	210

Table of Contents

Arithmetic Aggregate Functions	210
Uppercase and Lowercase Functions	210
Grouping for Summary Views	211
Complex Aliases	211
Nested <complex-alias> Elements	212
Extending Entities	213
Summary	214
Chapter 8: Accessing the Entities and View Entities	217
Setting-Up Our Playground	218
The Script Processor	218
The Generic Screen	219
Creating the Screen Widget	219
Creating the Form Widget	220
Creating the FreeMarker File	220
Publishing Our Generic Screen	222
Testing Our Playground	222
GenericValue Objects	223
Creating a Database Record	224
Why a Map is Used?	224
Updating Database Records	225
Deleting Database Records	226
Retrieving Database Records	226
Find Records by Conditions	227
Conditions	228
Comparison Operators (EntityComparisonOperator)	228
Condition Lists	231
Condition Joiners (EntityJoinOperator)	232
Tools for Common Styles of Conditions	238
Conditions Joined by AND	238
Conditions Joined by OR	239
Counting the Number of Records Retrieved	240
OFBiz Date Condition	241
Getting Related Records	243
One-to-One Relations	243
One-to-Many Relations	244
Utilities for Post-Query processing	245
Post-Query Ordering	246
Post-Query Filtering by Conditions	247
Post-Query Filtering by Date	248
Other Post-Query Filtering Methods	250
Using the Entity Engine Cache	251
Dynamic View Entities	252
Left Outer Joins with the Dynamic View Entity	256
Performing the Lookup	257

Table of Contents

EntityFindOptions	257
Paginating Using the EntityListIterator	258
Paginating through Party Records: A Working Example	258
Functions and Dynamic View Entities	263
Summary	263
Chapter 9: The Events	265
Java Events	265
Security and Access Control	266
User Logins are like Access Cards	267
Security Groups are like Access Levels	269
Security Permissions are like Individual Secured Areas	270
Security Permissions are Contained within Security Groups	270
User Logins are Assigned Security Groups	271
Dealing with Security in Java	271
Sending Feedback to the End-User	274
Conventions for Message Placeholders	274
Testing the Conventions	275
Handling Parameters	277
Accessing Localized Messages	279
Parameterizing Messages	281
Catering for Multiple Languages	282
Working with the Database	284
Summary	284
Chapter 10: The Service Engine	287
Defining a Service	288
Creating the Java Code for the Service	288
Testing Our First Service	289
Service Parameters	291
Input Parameters (IN)	291
Output Parameters (OUT)	293
Two Way Parameters (INOUT)	294
Special Unchecked Parameters	295
Optional and Compulsory Parameters	295
The DispatchContext	296
Service Security and Access Control	297
Calling Services from Java Code	299
Implementing Interfaces	303
Overriding Implemented Attributes	304
Synchronous and Asynchronous Services.	304
Using the Job Scheduler	305
Quickly Running a Service	307
Naming a Service and the Service Reference	307

Table of Contents

Event Condition Actions (ECA)	308
Service Event Condition Actions (SECAs)	308
Entity Event Condition Actions (EECAs)	310
Summary	311
Chapter 11: Permissions and the Service Engine	313
Simple Permissions	313
Two-Part Permissions and Special "_ADMIN" Permissions	316
Role Checks	317
Combining Multiple Checks	319
Nested Checks	320
Complex Permissions	321
Setting-Up Our Playground	321
Creating the Request and View Maps	322
Creating the Screen and Form Widgets	322
Creating the Entity PlanetReview	324
Defining Services that Require Complex Permission	325
Implementing Services that Require Complex Permission	325
Defining Permission Services	327
Implementing Permission Services	328
Playing with Complex Permissions	331
Complex Permissions and Simple Permissions cannot be combined	335
Summary	336
Chapter 12: Minilang	337
What is Minilang?	338
Tools to Code XML	339
Defining a Simple Service	339
Defining the Simple Service	340
Writing the Simple Method	340
Simple Events	342
Validating and Converting Fields	343
Validating a Simple Event Example	344
Checking Security in Minilang	348
Invoking from Minilang	349
Calling Services from Minilang	349
Calling Simple Methods	349
Calling Java Methods	350
Calling BeanShell	350
Minilang in Screen Widgets	350
Summary	351

Table of Contents

Chapter 13: Tying Up the Loose Ends	353
The OFBiz Look and Feel	353
Changing the Customer Facing Site	355
Changing the Back Office Screens	357
The Header	358
The Applications Bar (appbar)	358
The Central Area	359
The Footer	360
Using FreeMarker	360
OFBiz Transform Tags	362
<@ofbizUrl>	362
<@ofbizContentUrl>	363
<@ofbizCurrency>	364
Calling Java from FreeMarker	364
OFBiz Utilities	365
UtilMisc	366
UtilValidate	366
UtilDateTime	366
Debug	366
Outputting Different Formats	367
Outputting a PDF	367
Summary	370
Chapter 14: Tips and Techniques	373
Debugging Techniques	373
Logging	374
console.log	374
ofbiz.log	374
Configuring the Logs	374
Viewing the Logs through Webtools	375
Writing Information to the Logs	375
Logging Minilang	376
Debugging Java Code	378
Passing in the VM Parameters	378
Configuring the Remote Debugger	379
Managing a Project Using Subversion	382
Preparing the Repository and Creating the Project	383
Step 1: Checking Out the Latest OFBiz Code	384
Step 2: Making a Copy	384
Step 3: Importing Our Copy	385
Step 4: Branching ofbiz\current to Create the Trunk	387
Getting the Latest Features and Bug Fixes	388
Step 5: Updating OFBiz Current	388
Step 6: Merging Changes between Revisions of OFBiz Current into Our Project	391
Apache HTTP Server and OFBiz using mod_proxy_ajp	392

Installing the Apache HTTP Server	392
What is mod_proxy_ajp?	393
Configuring Apache HTTP Server to Use mod_proxy_ajp	393
Configuring OFBiz to Use the AJP Connector	394
Getting the Secure Pages to Work	395
Configuring SSL in Apache	395
Creating a Self-Signed SSL Certificate	396
Going into Production	396
Development Tips	397
Summary	398
Appendix A: Simple Method User's Guide	401
<simple-methods>	401
<simple-method>	401
Call Operations	402
<call-map-processor>	402
<call-service> / <call-service-asynch>	403
<set-service-fields>	404
Handling the Results of Service Calls	404
<results-to-map>	404
<result-to-field>	405
<result-to-request>	405
<result-to-result>	406
<call-bsh>	407
<call-simple-method>	407
<call-object-method>	408
<call-class-method>	409
Service Operations	409
<field-to-result>	410
Event Operations	410
<session-to-field>	410
Environment Operations	411
<set>	411
<clear-field>	412
<first-from-list>	412
Miscellaneous Entity Operations	412
<sequenced-id-to-env>	413
<make-next-seq-id>	413
Entity Find Operations	414
<entity-one>	414
<entity-and>	415
<entity-condition>	416
Sub-Elements	416
<condition-list>	416
<having-condition-list>	417
<select-fields>	417
<order-by>	417

Table of Contents

<entity-condition> complete example:	417
<entity-count>	418
<get-related-one>	418
<get-related>	419
<order-value-list>	420
<filter-list-by-and>	421
<filter-list-by-date>	421
Entity Value Operations	422
<make-value>	422
<clone-value>	423
<create-value>	423
<store-value>	424
<refresh-value>	424
<remove-value>	424
<remove-related>	425
<remove-by-and>	426
<set-pk-fields> / <set-nonpk-fields>	426
<store-list>	427
Control Operations	427
<iterate>	427
<iterate-map>	428
<check-errors>	428
<add-error>	429
<return>	429
If Conditions	430
<if-validate-method>	430
<if-instance-of>	430
<if-compare>	431
<if-compare-field>	432
Conditions:	433
Other Operations	434
<log>	434
<now-timestamp-to-env>	434
<now-date-to-env>	434
<set-current-user-login>	434
<calculate>	435
Index	437

Preface

Apache Open For Business or OFBiz as it is more commonly known, is an open source framework designed to facilitate the building of Enterprise Resource Planning (ERP) software. ERP is a general name for any system which attempts to integrate all business processes and underlying data into one single system. Indeed the OFBiz framework not only facilitates the building of your own custom software, but also comes packaged with many tools you would expect from an ERP system, and much more. The extent to which you wish to use these applications is entirely up to you and the needs of your business. Some businesses choose to use some or all of these components virtually straight out of the box. Others may spend time and money customizing these components or building new ones to suit their own needs and their own unique business processes. Since OFBiz is licensed under the Apache License Version 2.0, organizations can use, customize, extend, modify, repackage, and even resell OFBiz completely free of charge.

OFBiz is aimed primarily at ecommerce businesses, giving easily customizable tools such as a full Warehouse Management System (WMS), an accounting system and full order and product management systems. It even has a full front end, customer facing website and shopping cart with tools and features comparable to multimillion dollar websites such as Amazon, not to mention its own set of self maintenance and administrative tools. Out of the box, OFBiz is a multi-currency system working just as well with British Pounds, Euros, or any other currency as it does with US Dollars. It is multilingual and is able to display text in different languages depending on where in the world the user or customer is looking. It is so versatile it is not even tied to one database, and fully supports most well known databases.

The main reason for its versatility and size has been its open source model. OFBiz is truly a collaborative effort with a small number of committers who have volunteered to develop and maintain a code base supplied by both themselves and a growing community. Although documentation on the tools is often thin on the ground (this is mainly because of the speed at which the project and components evolve), there are free and active mailing lists set up that will become an invaluable learning tool and source of information as you progress with OFBiz. The OFBiz project employs the use of the well known JIRA application (a bug and issue tracking and project management tool – which is using the OFBiz Entity Engine, a major part of the framework). This allows developers and users to tell the community about any bugs they find in the software or request new features that they would find handy but perhaps don't have the resources to develop for themselves. Who knows? Once you have read this book you may even want to have a go at developing an outstanding issue or fixing a bug for the project yourself!

A Brief History of OFBiz

OFBiz was created in May 2001 by David E Jones and Andy Zeneski after they discovered that the problems they had tried to solve independently were similar and that existing solutions at the time were not sufficient. They realized that these problems could be solved using an Open Source model.

During the course of the next 5 years the OFBiz community grew and as such the number of organizations adopting OFBiz as their ERP system rose, and in January 2006 the project was accepted by the Apache Incubator Project Management Committee (PMC).

In December 2006 the Board of Directors voted to make Apache OFBiz a top-level project. Today OFBiz remains a highly active, community driven, not for profit project enjoying a growing membership and a wider adoption.

About This Book

The examples and tutorials featured within this book aim to, by the end of our journey, come together to produce a working web-based application. Various approaches will be explored during this project. Different combinations of techniques will be tried and evaluated, so we can learn when best to use them.

We will find that OFBiz is more than just a framework or toolkit. A good portion of OFBiz is comprised of ready-made functions and structures that easily lend themselves to being building blocks for ERP software, the application components. Whilst it is possible to use only the OFBiz framework to build a complete ERP application, it will be at least several magnitudes faster to build on top of these ready-made application components. We will study some of these applications while doing our project, which will also lead to a better understanding of some tried and tested best practices.

What This Book Covers

Chapter 1, using a Windows machine, guides us through downloading and installing the necessary software we need to obtain and run the OFBiz framework. We are shown how to create an Eclipse project and once running we are shown a few of the components as we place an order on the applications customer facing website and fulfill the order using the back office's Order Manager component.

In *Chapter 2* we learn the structure of OFBiz. We are introduced to the concepts of the framework, applications, and hot-deploy directories. We perform our first customization on an existing OFBiz component and finally create the structure of our own bespoke application.

In *Chapter 3* we take a look at how the output to the screen is constructed using the screen widgets. We start by creating a simple screen in our learning component, showing a basic output. By the end of the chapter we have learned how to create complex screens, made up of different sections.

In *Chapter 4* we study form widgets. We learn how they are used within screen widgets and can save us development time and effort by quickly producing XHTML forms so we can input information to the application.

In *Chapter 5* we complete our investigation into the presentation layer of OFBiz by learning how to use Menu-Widgets to navigate around our component. We also take more of a look at how FreeMarker can help us display more complicated screens.

In *Chapter 6* we re-visit the Control, learning more about how OFBiz makes use of the Front Controller pattern to configure the flow through our component in just one place. We learn how OFBiz handles different types of requests and we are introduced to the concept of security. By the end of the chapter, we have added "log in" functionality to our bespoke application and have seen how easy it is to force a request to be "secure".

In *Chapter 7* we move on to the concepts of the Entity Engine and learn how OFBiz employs the use of the Delegation Pattern to give us easy access to methods to persist data. We learn how OFBiz creates the database structure, adding fields, tables, constraints and indexes from definitions in XML files. We see how, by using View Entities, we can perform joins across tables allowing us to create complex queries. We are also introduced to the Webtools administration component of OFBiz and discover how to access the raw data through these screens.

In *Chapter 8* we are led through a series of examples designed to showcase data lookup and persistence techniques. We learn how to use the GenericDelegator's methods to lookup and manipulate the underlying data. We discover how using the Entity Engine Cache can massively improve performance by cutting down the number of database queries and learn how complex queries can be created on the fly by using Dynamic View Entity. Finally we learn how to use the EntityListIterator to efficiently paginate through large record sets.

In *Chapter 9* we take a closer look at Java events by learning a number of techniques vital to programming the flow of the application. We also take a look at how we can assign users permissions and how these permissions are checked within the Java methods.

In *Chapter 10* we next see another type of event and a very important one – the services. We learn about the advantages of the Service Engine and how it works, learning how to define and write services in Java. We learn the difference between invoking these services synchronously and asynchronously and how services can be scheduled using the Job Scheduler. Finally we learn how to trigger these services using ECAs (Event Condition Actions).

In *Chapter 11* we move on to study complex permissions, learning how to assign users granular permissions, and how simply these permissions can be checked from our services. We learn by example how to restrict users from viewing or inputting data depending on access rights whilst building up our bespoke application.

In *Chapter 12* we learn about the OFBiz Mini-Language. We see how we can write simple services and events in Minilang, and we learn when it should be ideally used. We see its versatility and see how widespread its concepts are used throughout the framework.

In *Chapter 13* we come towards the end of learning about the framework, we see how easy it is to change the look and feel of the component and study the structure of the existing screens. The chapter moves on to some more advanced FreeMarker techniques that are commonly used throughout all of the components. Finally using the production of a PDF as an example we see how to output different formats.

In *Chapter 14* we learn some real world developing techniques, including how to debug through the different parts and languages found within the framework. We see how to connect a remote debugger to the application and step through the Java code line by line using the Eclipse IDE. We next learn the concepts behind getting the latest bug fixes and features and merging these into our project using Windows tools enabling us to successfully work with the latest and greatest version on OFBiz. Finally we see learn how to run OFBiz behind the Apache HTTP Server allowing us to create a scalable architecture.

What You Need for This Book

Full download and installation instructions for all software and applications required to obtain and run OFBiz can be found in Chapter 1. These included:

- The Java Development Kit
- TortoiseSVN
- Eclipse IDE

At the time of writing, the latest stable version of OFBiz is 4.0. We will be working with that version, and not with the trunk version. In OFBiz speak (version control speak, rather), such a stable version is termed a "release branch"; the single trunk that forges ahead 24/7 is simply called the "trunk". Each stable branch can still move forward, due to the application of bug fixes. Hence, even release branches do go through revisions.

The trunk is the "latest, greatest and riskiest" state of OFBiz, where new features are added frequently, possibly along with new bugs. Such frequent addition of features do not allow for adequate testing. Before the dust can settle after a new addition by one contributor, more additions may come in from other contributors around the world. Scary as that sounds, this rapid and globally collaborative environment gives the OFBiz framework the fantastic advantage of being improved at a breakneck pace.

On the other hand, release branches like OFBiz 4.0 have had a "feature-freeze", and are invested with testing efforts focused on removing bugs and enhancing stability rather than adding new and untested features. Release branches are ideal for deployment in production environments (for real business use) where stability and ease of maintenance and support are vital.

Working with the trunk presents a moving target, a challenge that is best taken when we have some decent competency with the OFBiz framework as well as familiarity with recent OFBiz trends and changes.

This book may refer to existing code and also to line numbers within files. Bug fixes may therefore have an effect on these line numbers. Since they are only bug fixes, the line numbers should not drastically alter and a simple "Find" in the file will point you in the right direction.

The official policy on the frequency of releases (creation of new release branches) is published once per year, as defined in the Apache OFBiz Release Plan - General Release Policies <http://docs.ofbiz.org/display/OFBADMIN/Release+Plan#ReleasePlan-GeneralReleasePolicies>.

Who is This Book For

This book is intended for people wishing to understand and begin to customize the OFBiz framework. A basic level of Java is required and some understanding of Object Oriented Design (OOD) concepts would be beneficial. A basic grasp of Simple Query Language (SQL), although not strictly necessary, is certainly helpful to quickly understand some of the concepts in OFBiz.

Some fundamental skill with XML syntax is required, but only as far as is required to create well-formed XML documents.

The book will offer more hand-holding to MS Windows users. Users of other Operating Systems, such as Linux, should naturally be more adept at grasping the concepts laid out using Windows-specific examples and translating those examples into their respective OS's dialects.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " The <include-form> element includes a form named `FirstForm` residing in a file `LearningForms.xml`."

A block of code will be set as follows:

```
<request-map uri="OneForm">
<response name="success" type="view" value="OneFormScreen"/>
</request-map>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
server="default-server"
location="webapp/partymgr"
base-permission="OFBTOOLS, PARTYMGR"
mount-point="/partymgr"/>
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "We see two fields **First Name** and **Last Name**, and one **Submit** button.".



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.



Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to feedback@packtpub.com, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or email suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the Example Code for the Book

Visit http://www.packtpub.com/files/code/4008_Code.zip to directly download the example code.



The downloadable files contain instructions on how to use them.



Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to the list of existing errata. The existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide the location address or website name immediately so we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with some aspect of the book, and we will do our best to address it.

1

Getting Started with OFBiz

At the core of Apache OFBiz lies a powerful toolkit facilitating all aspects of software development—the **MVC (Model-View-Controller)** framework.

The MVC framework is an organizational structure for software. It covers three aspects: data (**model**), front-end (**view** or **user interface**), and logic (**controller**, the flow of the software). With this powerful toolkit alone, we can build a web-based software quickly. But OFBiz is more than that. OFBiz is also a collection or suite of ready-made **ERP (Enterprise Resource Planning)** components. Many of these components are so reusable and tried and tested, they have become quite **core** to OFBiz and to businesses that employ OFBiz.

The most prominent and polished aspects in OFBiz are ecommerce, product management, and order processing components. Aside from the commonly used ecommerce and related aspects, OFBiz also has a wide array of ready-made functions . These range from the commonly required accounting facilities to the more specialized manufacturing workflow.

In this chapter, we will be looking at:

- Obtaining OFBiz
- Compiling and running OFBiz
- Taking a quick first tour

Getting the OFBiz Code

Downloading OFBiz requires **Subversion (SVN)**—a version control system. For now, think of SVN as 'downloader software' that lets us download OFBiz, much like how 'browser software' (for example: IE, Firefox) lets us download TortoiseSVN (for Windows users) from <http://tortoisesvn.net/downloads>. TortoiseSVN is a graphical SVN client for Windows. It is neatly integrated with the ubiquitous and well-known Windows Explorer.

We only need the SVN client to talk to OFBiz project's SVN server in order to download a copy of OFBiz. We will use TortoiseSVN for the SVN client. Linux users can use the command line SVN client available from the Subversion website.

There is another means of downloading OFBiz – the Nightly Builds release. This is not the way recommended by OFBiz developers. The download method that uses version control software, such as SVN or CVS, is usually reserved for developers. Such a download method lets us check-out the source code that makes up the software (OFBiz in this case). In contrast, the download method meant for general (usually non-technical) audience involves downloading a single file that is usually an all-in-one installer and the software, which is to be installed. An example is the .msi installation file for TortoiseSVN.

OFBiz has so, far been targeted at developers, not the general folks who know or care little about programming software. We will now see how to download OFBiz using the recommended method.

Downloading and Installing SVN

We begin by obtaining the required 'downloader software' which is TortoiseSVN. TortoiseSVN is not the only Subversion client. Since TortoiseSVN carries a performance overhead and can slow Windows Explorer, many people choose to use the SVN from the command line using the Command Line client available from the Subversion website. For those already used to working with Eclipse, the Java Development Environment, there is also a handy plug-in called Subclipse. Full details of all Third Party clients and the Subversion command line installer are available on the Subversion Tigris.org website: <http://subversion.tigris.org>.

For the sake of the examples and tutorials, TortoiseSVN has been selected for its ease of use and simplicity, in spite of its performance overheads.

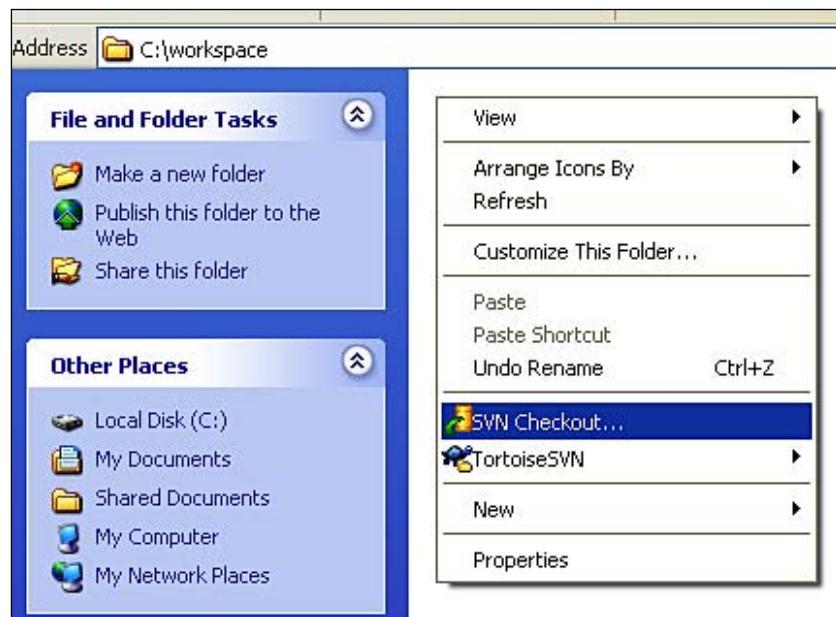
Downloading TortoiseSVN

We will first need to download TortoiseSVN <http://tortoisessvn.net/downloads>. There should be two versions: one for 32-bit operating systems and the other for 64-bit ones. Most of us will need the 32-bit version. If your Windows operating system has the term "x64" in its name, you'll need the 64-bit version. Download the appropriate one for your Windows operating system. Follow the on-screen instructions to complete the installation process.

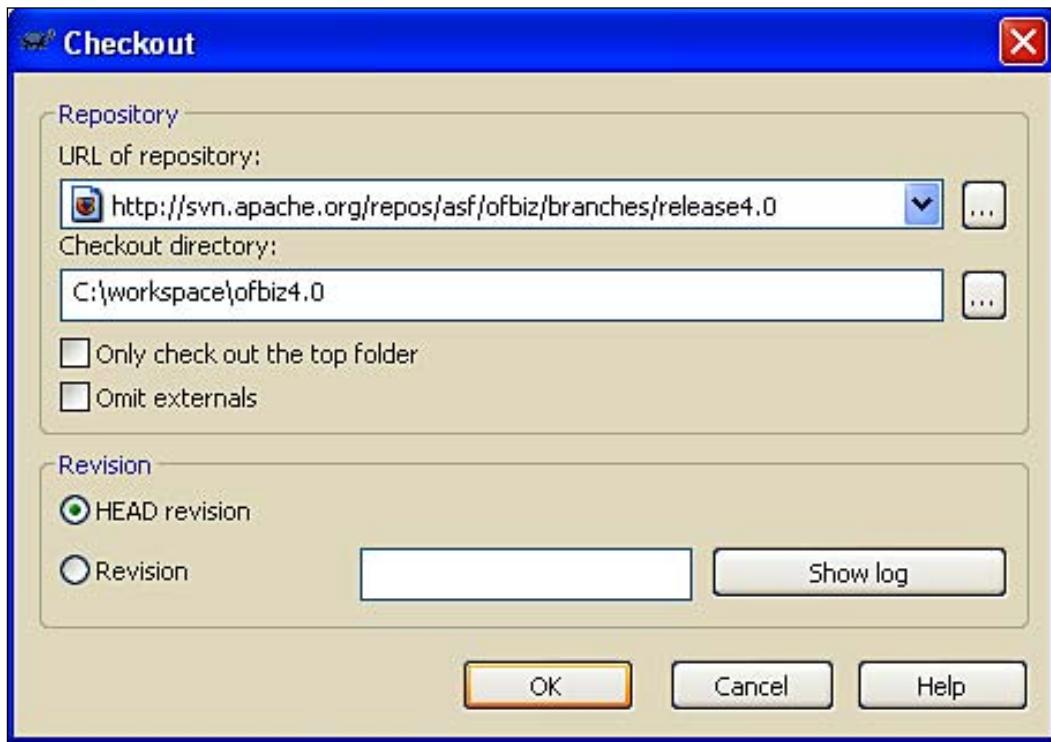
Using SVN to Get OFBiz

Create a folder where you want to store the OFBiz files. Let's call this the "OFBiz installation folder", hereafter referred to as \${OFBizInstallFolder}. Any folder will do because OFBiz does not need to be in any special place in your operating system in order to run. It is highly recommended, however, to store OFBiz in a location that has no spaces in its path, as this can sometimes cause strange errors. A good place could be C:\workspace\ofbiz4.0, since we are going to work with OFBiz 4.0 and eventually build an Eclipse workspace.

Go to the OFBiz installation folder using the Windows Explorer. Right-click on an empty area inside the Windows Explorer to bring up the "context menu".



Left-click **SVN Checkout** to bring up the **Checkout** dialog box. In the field **URL of repository**, enter `http://svn.apache.org/repos/asf/ofbiz/branches/release4.0`.



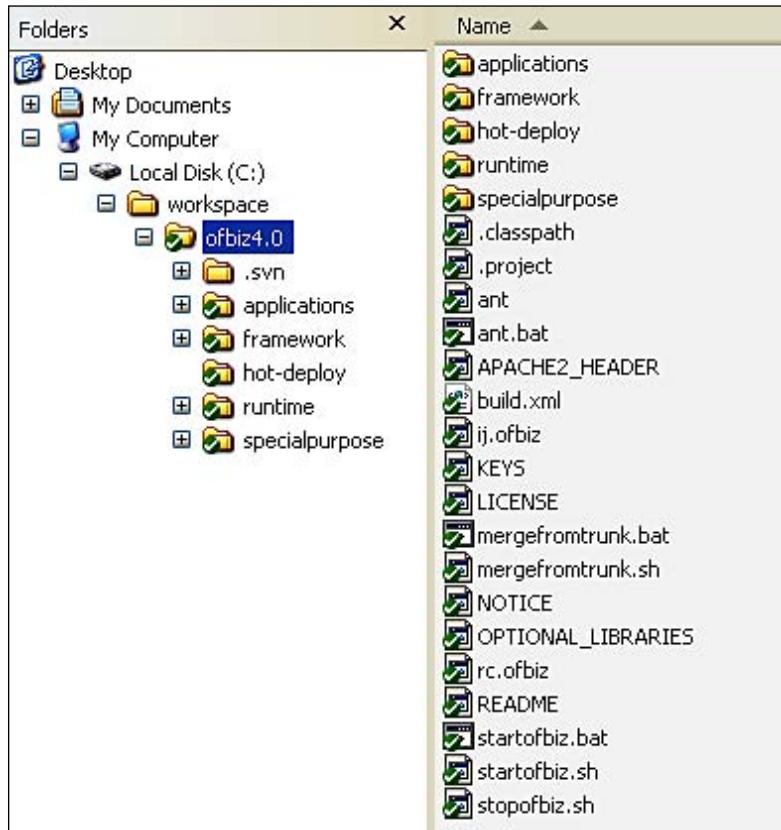
If the folder `ofbiz4.0` does not exist, you will first be informed and asked if you wish to create it.

You should now see the files being checked out in the SVN information box. Once you complete the final message in the box will say **Completed** and give the revision number of the version that was checked out.

At the time of writing, the total size of the checkout was 25.69 MB. It should be noted that this includes all of the source code and libraries that OFBiz needs to run. Needless to say, the length of time it takes to check-out the code depends on the speed of the connection.

Our OFBiz Workspace—First Look

When the check-out process is completed, we will see our OFBiz SVN workspace.



Depending on your Windows Explorer settings, you may or may not see hidden folders and files. You may see grayed out (hidden).**svn** folder which contains the SVN control files, something we should not have to bother with.

As we proceed and begin to change files, you may see the green ticks change to:



This means the folder (or the contents of the folder rather) has changed locally since the last update (or initial checkout). For now we do not need to worry too much about the workings of SVN. Since we will not be committing the changes back into this SVN server, we can ignore these icons.

Installing the Java Development Kit (JDK)

Before we can run OFBiz, we will need to compile it.

OFBiz is written largely in Java, and the framework is entirely in Java. Code written in Java needs to be pre-parsed and pre-compiled before it can be executed. To compile OFBiz, we will need the Java Development Kit (JDK) 5.0 [http://java.sun.com/javase/downloads/index_jdk5.jsp.] The latest JDK 5 version at the time of writing is JDK 5.0 update 13. That is the version we will be using in this book. Later revisions (or "updates") of JDK 5.0 should also work the same.

Strictly speaking, OFBiz 4.0 only requires Java 1.4. However, since the current ("trunk" branch) OFBiz development uses Java 5, and the next OFBiz release will likely use Java 5, we should get used to Java 5 now. Note that full Java 5 syntax is not yet supported in BeanShell, a scripting language will be used throughout this book, so we will still be doing Java 4 syntax when writing in BeanShell.

Downloading JDK 5.0

On the JDK 5.0 download page, we should see at the top of the download items list the version for **Windows Platform**. We want the **Windows Offline Installation**, Multi-language download (at 51.42 megabytes).

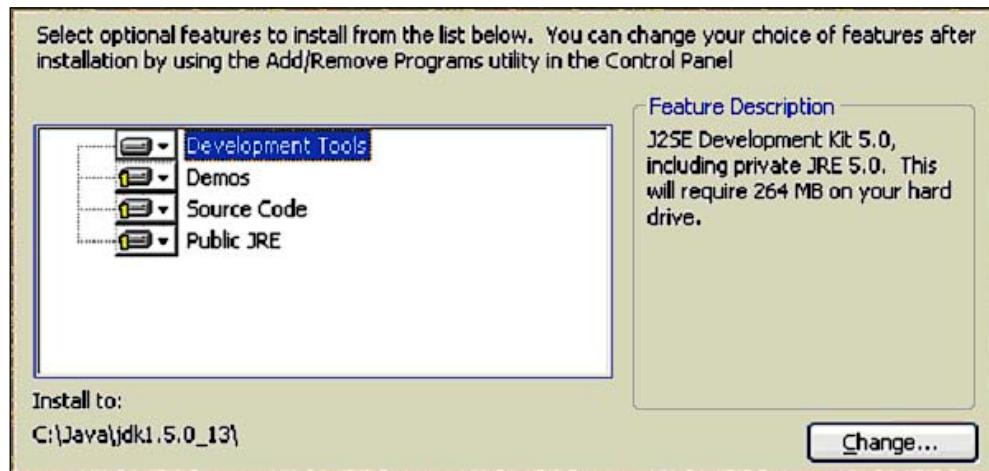
The technical name for JDK 5.0 is really JDK 1.5, as can be seen from the filename **jdk-1_5_0_13-windows-i586-p.exe**. JDK 1.5 came in after JDK 1.4. JDK 1.5 was named JDK 5.0, possibly for marketing purposes. More details about the change of naming convention are at <http://java.sun.com/j2se/1.5.0/docs/relnotes/version-5.0.html>.

The version we're working with is for the operating system MS Windows 32-bit. If you're using a 64-bit operating system, you can try the version under **Windows x64 Platform**. The filename **jdk-1_5_0_13-windows-amd64.exe** suggests it may only be for an **AMD 64-bit** computer, but that binary should also be compatible with **Intel 64-bit** computers.

In case you encounter any problems with the download of the JDK 5.0, see the Sun Download and Installation Notes at <http://java.sun.com/j2se/1.5.0/install.html>.

Installing JDK 5.0

Double-click on **jdk-1_5_0_13-windows-i586-p.exe** to start the installer. In the installation setup screen, change the **Install to** value to **C:\Java\jdk1.5.0_13**. By default, the Java Installer may suggest to install to **C:\Program Files**. Installing to the default location has been seen to cause errors, particularly in the **RMI (Remote Method Invocation)** code on startup. This is due to the white space between **Program** and **Files**. Leave the feature **Development Tools** selected, and leave all other features unselected. For our purposes here, we only need the JDK and the private JRE 5.0. Proceed to complete the installation process.



Once the installation of the JDK is complete, we need to check that the environment variable **JAVA_HOME** is set correctly. To do this, click **Start Button | Control Panel | System** and select the **Advanced Tab**

Towards the bottom of the dialog box, you should see the **Environment Variables** button. There are two things to check here, first ensure that there is a variable set up called **JAVA_HOME**, pointing to the location **C:\Java\jdk1.5.0_13** and then ensure that there is an entry under the **PATH** variable pointing to **C:\Java\jdk1.5.0_13\bin**. Entries in the **PATH** must be separated by a semi colon(;). If in the unlikely event these don't exist, they must be added manually, preferably as System Variables.

Downloading OFBiz Ready to Launch

As an alternative to SVN, if you prefer to see how OFBiz works, you may use the **Nightly Builds** page at <http://build.hotwaxmedia.com>. Then simply follow the instructions on this page. Please remember that this book is intended to be used with Release 4.0

Setting Up an Eclipse Project

Eclipse, like OFBiz, is an open-source project. It is an **IDE (Integrated Development Environment)** used by Java Developers all over the world. It is available free of charge for Windows, Linux, and Mac OS X and is the most commonly used Java Development Environment available.

To download Eclipse, head to the Eclipse project download page at <http://www.eclipse.org/downloads> and select the option **Eclipse IDE for Java Developers**. You will be offered a preferred mirror site to download from. Simply select it and save the zip file to the root of your C: drive.

At the time of writing, the file was around 80MB. Unlike the TortoiseSVN program we installed earlier, this program does not need to be installed. The zip file can be de-compressed and the executable (program) file can be run straight away. Once the file has downloaded, right-click on it, select **Extract All**, then change the suggested location to C:\ and press **Next**.

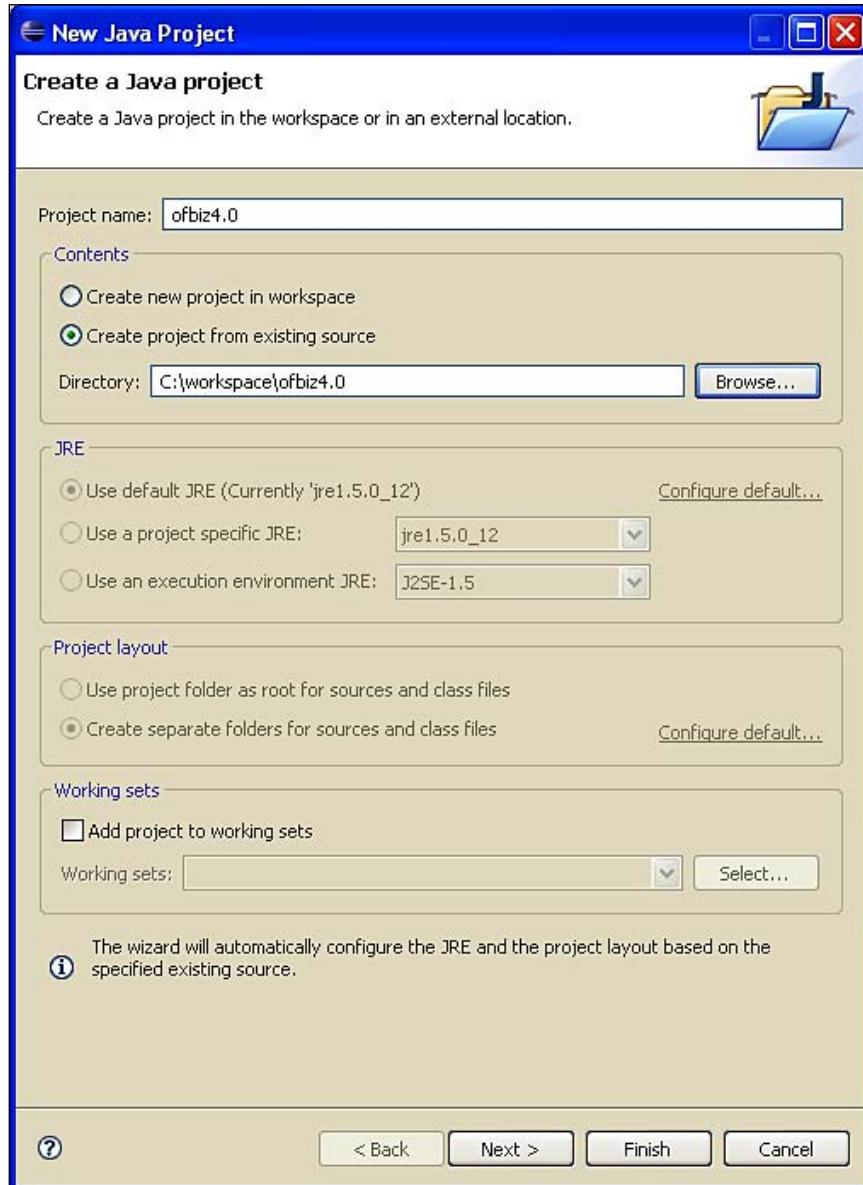
This will place a new folder on your hard drive called C:\eclipse. This is all the installation that eclipse needs to run. If you go into the C:\eclipse folder you will see the spherical Eclipse icon. For ease, you could create a shortcut by right-clicking and selecting **Create Shortcut**, then drag-and-drop this onto the desktop. Double-click it to start Eclipse.

When Eclipse is run for the first time, something like the following appears:



Since the OFBiz code has already been checked out to C:\workspace, make this location the Eclipse workspace and select the **Use this as the default** checkbox and click **OK**. Unless you want to explore the features of Eclipse before you begin, close down the welcome tab and click **File | New Java Project**.

In the **Project name** text field, type **ofbiz4.0** and change the **Contents to Create project from existing source**.



Click **Finish** to continue and the project will be created. By default, Eclipse will show the *Package Explorer* view, with all the Java source folders (packages) appearing on the left-hand-side of the screen. Experienced Java developers will be used to working like this, however, because of the amount of different files we will be looking at throughout the course of this book, it is easier to change the View to the *Navigator* view, by clicking **Window** from the main bar and selecting **Show View** and **Navigator**. This view is the file structure of your project and looks similar to the layout we saw in Windows Explorer, but now navigating around the project and opening, saving, and closing the files is much simpler. There is a free plug-in for Eclipse to turn it into a FreeMarker editor (<http://freemarker.sourceforge.net/editors.html>). This is highly recommended to aid working with FreeMarker files.

We can even compile and build our project from within Eclipse. Before compiling, it may be worthwhile to check that Eclipse is using the JDK we installed; it may try and use a JRE that was previously installed on the system. To check this, go to **Window | Preferences** and search for **Installed JREs**, and check that it's using the correct one. To compile the project, find the `build.xml` file (the one with the picture of the ant on it) in the root of the project, right-click and choose **Run As | Ant Build**. You've just compiled all the code and built an OFBiz project for the very first time.

There is one more thing to do in Eclipse that will make life somewhat easier. Return to the **Window** option from the top menu bar, choose **Show View**, and this time select **Ant**. On the right-hand side you should see the **Ant** box. This will be empty at first, but you can quickly add the main **build** file by going back to the **Navigator** box and dragging it over. With this view open while you are working, it is quick to build the whole project, or if you choose you can add other component's `build.xml` files from within the project to the **Ant** box. Although we will not be running the project from within Eclipse, individual components can be navigated to and compiled quickly and independently of the main project.



Alternatively, you may right-click on a build.xml file and just use **Run** as the option in the context menu and then use the first available option. You will then be able to use the specific icon **Run** in the menu bar to easily set and run builds.

Using Derby—the OFBiz Stock Database

OFBiz comes pre-packaged with Apache Derby, an open source **Relational Database Management System (RDBMS)** implemented entirely in Java. The default OFBiz configuration uses Derby, so there's nothing we need to do to connect to a database. How to change the database settings and which database OFBiz uses are well documented in the OFBiz Entity Engine Configuration guide at the OFBiz website at <http://ofbiz.apache.org/docs/entityconfig.html>. We will be investigating the Entity Engine in depth, for now think of this as the database.

Compiling OFBiz and Loading the Data

Expand the OFBiz Main Build entry in the Ant View Box in Eclipse by clicking on the + sign. We can now see the first **target** is in blue and called **build[default]**. The default target is the one that is executed when you double-click (or right-click and **Run As**) on the **OFBiz Main Build** entry. This default build target reads and executes each component's `build.xml` file. Quite simply, the full build process for a component is:

- delete the build folder containing existing compiled code
- re-compile the java code, "including any changes made since the last compilation"
- re-create the jar
- place in a new build folder

However, to begin with, we want to go a step further than this. Ant targets have been included in the main script that performs the build process, and allows the database structure to be automatically created, including all necessary database indices and constraints. OFBiz then loads some "seed data" that it needs to start up. An example of an important piece of "seed data" that is loaded into the database is information on the administrator user that we will need to login to the OFBiz components. If we were to start OFBiz without loading this, we will not get very far as we cannot login. Make sure that the **Console** window is open and in view at the bottom of Eclipse (it is open by default), then double-click the run-install target in the **Ant** view window.

Verifying the Installation Process

Debug messages should now be scrolling away within the Console window. They may be scrolling too fast to be seen by the naked eye. Do not worry if they are, this information is also being saved to the `ofbiz.log` found in `${OFBizInstallFolder}\runtime\logs`.

The Compilation Portion

The first half of the log shows the compilation process itself. It should end with `[echo] [build] ===== Done Building (Compile) =====`. There should be six warnings, which are due to some outdated code in the "order" component. They can be seen right above the line that says:

```
[jar] Building jar: ${OFBizInstallFolder}\applications\order\build\lib\ofbiz-order.jar
```

These warnings can be safely ignored. Search for the keyword **error** to spot compilation errors. Look for unanticipated warnings. Any signs of those will mean the compilation process was not entirely successful, and has to be redone. The compilation logs will not be written to the `ofbiz.log` file, since the application has not yet been started.

The Data Loading Portion

The next half of the log shows the data loading process that loads the start-up data for OFBiz. This is the lengthy part of the **ant run-install** process and, depending on the speed of your computer, this part of the process could take five or even ten minutes. In comparison, the compilation process is very short, merely one to two minutes long. On successful completion of the build and seed data install, you should see a similar message to this:

BUILD SUCCESSFUL
Total time: 8 minutes 33 seconds

There are two kinds of anomalies to watch out for whose keywords are **WARN** and **ERROR**. The absence of **ERROR** should mean a successful data-loading process. Unfortunately, as the log message format is right now, there is no easy way to spot the truly problematic **WARN** logs.

Readers with some ability with Linux and the grep utility may find it useful to filter out lines containing the regular expression **WARN] .*has no table in the database**. Even once these messages have been removed, we may still see other **WARN** logs that are problematic but not fatal show-stoppers. The stock OFBiz is still a "work-in-progress", but is not difficult to enhance quickly. As we become more competent with OFBiz, we may be able to help enhance OFBiz over time and improve the error-reporting format for the data-loading process.

Some Pre-Ignition Processes we won't do Twice

Because the initial data-loading process can take so long, we should try to avoid doing it more than once. Some folks may have faster computers that can do it all in a couple of minutes, but even these folks need to consider the typical cycle of activity when dealing with software development:

- Change (the software code)
- Test (the new changes)
- Reset (the data to prepare for next test case)
- Test (the next test case)
- Revise change (if we encounter bugs)

The cycle goes on until we have confirmed that the new change we made satisfies our requirements and has been tested enough for us to be reasonably sure it is bug-free.

It's like performing a science experiment or trying to perfect a food recipe. After every attempt or experiment, the cooking or science apparatus need to be washed and "reset" to a clean state so that the results of subsequent attempts or experiments will not be tainted by prior ones.

In software testing, every test will change the data in some way, unless the test involves mere data retrieval. Three of the four database activities CRUD (create, retrieve, update, delete) will change the data. Even for mere retrieval tests, the software may need to write "retrieval logs", which will translate into "create-related" tests.

As can be seen, it can take five to ten minutes to wash the cooking apparatus and start over (data-loading from scratch takes that long). Perhaps buying an automatic dish-washer will cut that "reset" time down to two minutes. Now imagine if we could "reset" in just seconds.

It must be noted that any changes to the seed data itself will require a new data-loading process from scratch.

Backing Up the Derby Data Files

The Derby data files are in `${OFBizInstallFolder}\runtime\data\derby`, contained in a folder named `ofbiz`.

Go to `${OFBizInstallFolder-}\runtime\data\derby` and zip up the Derby data files into a single archive which you can call `ofbiz_4_0_r589272.zip`. Keep the archive safe somewhere. Removing the Derby data files in `${OFBizInstallFolder}\runtime\data\derby` and replacing them with those in the archive will quickly "reset" our OFBiz data state.

For now, we just back up the Derby data files and move on.

Running OFBiz

Now that we've gone through the necessary pre-startup steps, it's almost time to start OFBiz. But before we start we'll need to check our available memory.

Allocating Memory for OFBiz

To check our memory availability, bring up the Windows Task Manager by pressing `Ctrl+Shift+Esc`. Click on the tab named **Performance**.

Another way to access the Windows Task Manager is by right-clicking on an empty space on your task bar (try the space around your clock in the bottom-right of your desktop), and then left clicking on **Task Manager** select the **Performance** tab.

Physical Memory (K)	
Total	2094952
Available	1032348
System Cache	938228

The remaining available physical memory we have is stated under the Physical Memory section, beside the label Available. It is in units of kilobytes (or "K" in short). OFBiz's memory switches are in units of Megabytes. 1 megabyte is about 1000 Kilobytes (slightly more, actually). In our system, we have about 1,000 Megabytes (1 Gigabyte) available.

The amount of memory used by OFBiz is determined by some switches in the file \${OFBizInstallFolder}\startofbiz.bat (Linux users can see the .sh equivalent). Edit that file, and look for the following line:

```
%JAVA_HOME%\bin\java" -Xms256M -Xmx512M -Duser.language=en
-jar ofbiz.jar > runtime\logs\Console.log
```

The number 256M (representing 256 Megabytes) right after -Xms is the minimum amount of memory that OFBiz will use. OFBiz will set aside that amount of memory when it starts. The number 512M right after -Xmx is the maximum amount of memory OFBiz will use. OFBiz will grow in memory usage, if necessary during the course of operation and depending on memory availability, up to this size.

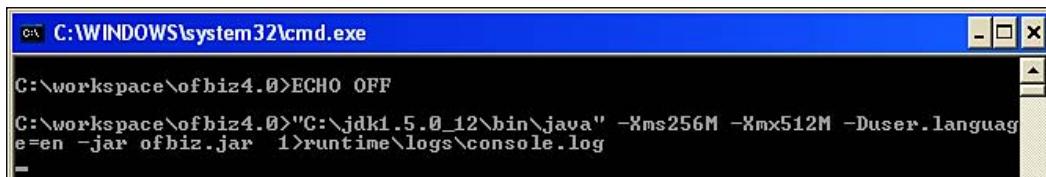
The minimum size is set aside during start-up, so we should set this figure to use an amount of memory that we are confident will always be available. In our example, we have 1 GB of memory available, and should set OFBiz's minimum memory usage to less than that, probably **512M** or less. Also try to make sure the maximum size is not larger than our available memory. **768M** is a safe bet in our case.

If you have less than 128 megabytes of physical memory, OFBiz will run awfully slowly. It will still run because Windows (and other operating systems) has the concept of "virtual memory", which really is your hard disk space. Hard disk access is slower than RAM (physical memory chips) access.

For production with Linux servers it is advisable to assign at least 2 GB of RAM to OFBiz. There is no reason why you cannot assign the minimum amount and the maximum amount of memory OFBiz will use both to 2048 MB. This will ensure that 2 GB is assigned at all times. Windows is not able to allocate more than 1.5GB to a JVM instance unless we use a 64 bit version.

Starting OFBiz and Verifying the Start-Up Process

Now, we start the engines! Navigate to the \${OFBizInstallFolder} in Windows Explorer and double-click the file **startofbiz.bat**. An un-editable command prompt window will appear. And OFBiz will startup.



```
C:\WINDOWS\system32\cmd.exe
C:\workspace\ofbiz4.0>ECHO OFF
C:\workspace\ofbiz4.0>"C:\jdk1.5.0_12\bin\java" -Xms256M -Xmx512M -Duser.language=en -jar ofbiz.jar 1>runtime\logs\console.log
```

As we will be frequently starting the OFBiz application, it is a good idea to make the start process quick to access. The Windows Task Bar is an ideal place to click from, as no matter what you are working on, you can start OFBiz. This can be done by right clicking on the **startofbiz.bat** file, creating a shortcut and then dragging and dropping the shortcut to the unlocked taskbar.



The only reliable way to confirm that OFBiz has finished its start-up process is to look at the file \${OFBizInstallFolder}\runtime\logs\console.log and check for the following lines.

Connector AJP/1.3 @ 8009 - not-secure [org.apache.jk.server.JkCoyoteHandler] started.

Connector HTTP/1.1 @ 8080 - not-secure [org.apache.coyote.http11.Http11Protocol] started.

**Connector TLS @ 8443 - secure [org.apache.coyote.http11.Http11Protocol] started.
Started Apache Tomcat/5.5.20**

To make sure that OFBiz was started without any problems, search the **console.log** for keywords **Exception** and **Error**. The absence of those should mean that OFBiz started without any problems.

Seeing the Logs in Real Time

It may, at times, be convenient to see the logs in real time, rather than locating the logs to see what happened after the event. This can be done by removing the `> runtime\logs\console.log` part of the line in the `startofbiz.bat` file. The logs will no longer appear in `console.log` and will instead scroll inside the command prompt. The logs will still be recorded in the `ofbiz.log` file.

Possible Problems with Start-Up

OFBiz listens on a number of ports. Think of "ports" as a house or door number. Server programs receive incoming requests from "client" components (each one operated by some end-user). There can be many programs running on one computer, with each program receiving incoming requests from any number of "client" components. Therefore, each incoming request will need to specify the port it is "knocking on", so that the server computer will know where to direct the request.

One possible problem during start-up is that any one of the ports OFBiz listens on are already occupied by another program. In this case, `${OFBizInstallFolder}\runtime\logs\console.log` may contain the following lines:

```
[Http11BaseProtocol.java:140:ERROR] Error initializing endpoint  
java.net.BindException: Address already in use: JVM_Bind:8080
```

which indicates port 8080 is already in use. Full instructions on changing the ports to which OFBiz listens can be found in the Apache OFBiz Production Setup Guide at <http://docs.ofbiz.org/display/OFBTECH/Apache+OFBiz+Technical+Production+Setup+Guide>.

Some commonly used programs including Skype, MSN Messenger, and Apple iTunes often use ports 80 and 1099 which will cause problems with start-up. If problems are experienced, these programs should be stopped.

Switching Off OFBiz

Some customizations require that we first shut down OFBiz. Many other types of customizations don't. As we progress on our journey it will become clear which require a restart, which will require a restart and compile, and which will require nothing more than a refresh of the browser.

Shutting down OFBiz is as simple as focusing on (selecting, clicking on) the OFBiz start-up shell, and pressing *Ctrl+C*. We'll see the shell display a prompt **Terminate batch job (Y/N)?**. Reply **Y**. In \${OFBizInstallFolder}\runtime\logs\console.log, we'll see a series of shutdown logs:

```
[ ContainerLoader.java:93 :INFO ] Shutting down containers
[ ServiceDispatcher.java:176:INFO ] De-Registering dispatcher: example
[ ServiceDispatcher.java:176:INFO ] De-Registering dispatcher: partymgr
...
...
[ ServiceDispatcher.java:176:INFO ] De-Registering dispatcher: RMIDispatcher
```

Any shutdown errors (these are very rare) will be shown in the `console.log` as well.

It should be noted that if you close the shell window by pressing the top-right **X**, as you would with any other window, OFBiz will shutdown.

Our First Tour of OFBiz

It's time to step on the gas pedal and take in the sights! We'll be zipping through some often used (and most developed) areas of OFBiz, touring from the perspective of an end-user (not from a developer or programmer). Our aim is to take a quick look at what OFBiz is capable of, as well as to learn the common pattern of accessing OFBiz webapps (such as "ecommerce" and "ordermgr").

The term **webapp** will be explained further in Chapter 2. End-users access OFBiz through its webapps. The term component, which is related to webapps, will also be discussed in Chapter 2.

Accessing OFBiz

OFBiz is accessed through the web-browser and works just as well through Internet Explorer as it does Firefox. By typing an address into the URL we are sending a request to the application.

Firing a request to OFBiz involves giving at least three pieces of information, with an optional fourth:

- server name (where to locate the server computer)
- port (which port on the server computer to "knock on")
- OFBiz webapp name (which OFBiz webapp to call on)
- Request (like an instruction, or request for service)

The **server name** in our case will be **localhost**.

The **port** will be 8080 for protocol **http** (unsecured access), and 8443 for **https** (secured access). Therefore, in this book, all URLs to OFBiz will start with `http://localhost:8080/` or `https://localhost:8443/`.

The fourth piece of information is optional in most cases. If missing, this will default to a request for some welcome or "main" screen, if such a screen is set up. From here on, when we talk about firing a **request**, that request will be just this piece of information, not including the other three pieces of information. For example, firing an http request `ecommerce/index.html` will mean an URL of `http://localhost:8080/ecommerce/index.html`, whereas an https request will mean the URL starts with `https://localhost:8443/`.

Often, we will say "fire a request **<request string>** to webapp **<webapp name>**". For example, firing a request `index.html` to webapp `ecommerce` is the same as firing a request `ecommerce/index.html`. When we are to fire an empty request string to some webapp, say `ecommerce`, we will say "fire an http request to webapp `ecommerce`", which should be taken to mean an URL of `http://localhost:8080/ecommerce`.

Fire an http request to webapp `ecommerce` and note how we are redirected to the welcome screen at `request control/main`. Note how the two requests show the same screen.

To access the running OFBiz ecommerce application, open an internet browser in the same computer that is running OFBiz. Assuming that we were able to use the default OFBiz ports, a typical end-user request to OFBiz looks like `http://localhost:8080/ecommerce`, which calls on the webapp `ecommerce` in OFBiz.

From here on, instead of saying "open an internet browser and enter a URL address of whatever to fire a request to some OFBiz webapp", this book will simply say "fire a request **<request name>** to webapp **<webapp name>**".

Exploring the Webapp "ecommerce"

Fire an http request to webapp ecommerce. The main screen of the webapp "ecommerce" shows the list of **Featured Products** available for ordering via the OFBiz ecommerce site.

The screenshot shows the main interface of the Open For Commerce webapp. At the top, there's a header with the Apache OFBiz logo and the text "Open For Commerce Part of the Open For Business Family of Open Source Software". On the right, it says "Shopping Cart is empty [View Cart] [Quick Checkout]". Below the header, there are several navigation links: Login, Contact Us, Main, Quick Add, Order History, Shopping Lists, Requests, Quotes, Profile, Language (set to English), Cart Summary (empty), and Special Offers.

Featured Products:

- Financial Account Activation**: Balance Account Activation FA-001 Your Price: From \$1.00. Choose Variation... button.
- Gift Card Activation**: Give the perfect gift! GC-001 Your Price: From \$1.00. Choose Variation... button.
- Gift Card Reload**: Add more money to your card! GT-002. Choose Amount... button.
- Round Gizmo**: Round Gizmo with lights - Usually ships in 15 Days! GZ-2644 List Price: \$40.00 **On Sale!** Your Price: \$30.40 Save: \$9.60 (20%). Add to Cart button.
- Configurable PC**: Configurable PC PC-001 Your Price: \$50.00. Configure... button.
- Tiny Chrome Widget**: Tiny Chrome Widget - Usually ships in 2 Days! WG-5569 List Price: \$60.00 **On Sale!** Your Price: \$48.00 Save: \$12.00 (20%). Add to Cart button.
- Giant Widget**: Giant Widget with Wheels Sizes Available: 3-Wheel, 4-Wheel WG-9943 Compare At: \$92.00 List Price: \$50.00 **On Sale!** Your Price: From \$40.00 Save: \$110.00 (20%). Choose Variation... button.

On the left sidebar, there are sections for Choose Catalog (Demo Catalog, Change), Search Catalog (Advanced Search), Browse Categories (Account Activation, Configurable PCs, Gift Cards, Gift Card Reload, Gift Card Purchase, Widgets (english), Large Widgets, Small Widgets, Other Mini Widgets, Widgets en Micro Widgets (english), Gizmos (english), Large Gizmos, Small Gizmos), Mouse Hand Poll (Right Hand), and Browse Forums/Browse Content (Gizmos, Policies, Widgets).

Let's Buy Something

Buy **4 Round Gizmo** (fourth item down the Featured Products list, product ID GZ-2644) by entering **4** into the text field beside the item and clicking the **Add to Cart** button beside it. We'll see the cart (top-right of screen) now contains four items.

The screenshot shows the Cart Summary page. It displays a table with the following data:

# Item	Subtotal
4 Round Gizmo	\$50.00
Total: \$45.00	

Below the table are buttons for View Cart, Checkout, and Quick Checkout.

Click on **Checkout** in the **Cart Summary** screenlet (a screenlet is simply a sub screen, a part of the main screen). Because we're currently not logged in, OFBiz brings us to the "Login" screen. If this is the first time that you have accessed this login screen, you may notice that the URL jumps from being `http://localhost:8080` to `https://localhost:8443` and a warning will appear asking you if you wish to accept the certificate, in spite of a domain name mismatch. Depending on your browser, click **Yes** or **OK** to accept the temporary SSL certificate. This acceptance only applies for the "localhost" address and is perfectly safe.

For now, we do a **Checkout Without Login**. Click the **Quick Checkout** button.



The **Quick Checkout** button brings us to the "Shopper's Personal Info" page. In the **Name Phone and Email** section, enter any value for **First name**, **Last name**, **Home phone**, and **Email address**, fields that are mandatory.

Basic Information	
Name Phone and Email	
First name	<input type="text" value="OFBiz"/> *
Middle initial	<input type="text"/>
Last name	<input type="text" value="Researcher"/> *
[Country Code] [Area Code] [Contact Number] [Extension]	
Home phone	<input type="text"/> - <input type="text"/> - <input type="text"/> - <input type="text"/> *
Business phone	<input type="text"/> - <input type="text"/> - <input type="text"/> - <input type="text"/>
Email address	<input type="text" value="someaddr@somewhere.com"/> *

In the **Shipping Address** section, simply fill in some values for the mandatory fields (marked by *).

Shipping Address	
To Name	<input type="text" value="Gift Recipient"/>
Attention Name	<input type="text"/>
Address Line 1	<input type="text" value="Somewhere"/> *
Address Line 2	<input type="text"/>
City	<input type="text" value="A City"/> *
State/Province	<input type="text" value="Alaska"/> *
Zip/Postal Code	<input type="text" value="123456"/> *
Country	<input type="text" value="United States"/> *

Check the checkbox labeled **Billing address is the same as the shipping address**.

<input checked="" type="checkbox"/>	Billing address is the same as the shipping address
-------------------------------------	--

Click the **Continue** button at the bottom-center of the screen to confirm the shopper's (that's us!) personal info and to continue to the next step.

This brings us to the "Final Checkout Review" screen. In the **Payment Information** section, select **Offline Payment: Check/Money Order**.

Order Information	
Status	Not Yet Ordered
Payment Information	
Select Payment Method	<input type="text" value="Offline Payment: Check/Money Order"/> *
<input type="checkbox"/> Check If You Have A Gift Card To Use Today	

In the **Shipping Information** section, select any shipping method desired (try the first one so we're playing with the same options).

Shipping Information	
Destination [1]	To: Gift Recipient Somewhere A City, AK 123456 USA
Method	
<input type="radio"/> UPS Guaranteed Next Day - \$19.80	
<input type="radio"/> UPS Air - \$10.60	
<input type="radio"/> UPS Ground - \$5.80	
<input type="radio"/> USPS Express - Calculated Offline	

Finally, take one last look at our order before we submit the order.

Order Items					
Product	Qty Ordered	Unit Price	Adjustments	Subtotal	
GZ-2644 - Round Gizmo [Quantity: 100] [Weight: 7] Adjustment: Promotion	4	\$38.40	(\$103.60)	\$50.00	
				(\$103.60)	
					Subtotal \$50.00
					Promotion (\$5.00)
					Shipping and Handling \$19.80
					Sales Tax \$0.50
					Grand Total \$65.30
Submit Order					

Submit the order by clicking **Submit Order**.

Our order will be processed and created. We are brought to the **Order Confirmation** screen next, which tells us that our order was successfully created.

Order Confirmation	
NOTE: This is a DEMO store-front. Orders placed here will NOT be billed, and will NOT be fulfilled.	
Order #WS10000 Information <hr/> Name OFBiz Researcher <hr/> Status Created <hr/> Date 2008-05-29 23:35:07,609	Delivery Information <hr/> Destination To: Gift Recipient [00001] Somewhere A City, AK 123456 USA <hr/> Method UPS Guaranteed Next Day
Payment Information <hr/> Offline Payment <hr/> Please Send Payment To: Company XYZ 2003 Open Blvd Open City, CA 999999 USA <hr/> Be sure to include your order #	Splitting Preference Please wait until the entire order is ready before delivering. <hr/> Gift? This order is not a gift.

We can take a look at the items we have successfully ordered further down the screen.

Exploring the Webapp "order"

After buying something from the ecommerce site as an anonymous shopper, we will now log in as a back office member of staff to process the newly created order.

Fire an http request to webapp `ordermgr`. We'll be brought to the login screen of webapp `ordermgr`, where we log in with username of **admin** and password of **ofbiz**.

The main screen of the webapp `ordermgr` is the **Order List** screen, which lists existing orders. So far, there should only be one order, which we created in the last section in ecommerce. Click on that order to view its details.

Receiving Payment

Click on the **Receive Payment** button in the **Payment Information** screenlet.

Payment Information

Offline Payment

Max Amount: US\$65.30

Receive Payment **Cancel**

There will be many options for receiving various types of payments, such as through credit cards or paypal or gift cards (vouchers) or cash/check. Let's choose to receive cash. Enter **65.30** (\$65.30) in the cash **Amount** field, and any random string (say **cash123456**) in the **Reference** field. Then click the **Save** button to confirm the receipt. You may notice that the status of the order has now moved from **Created** to **Approved**.

An **Offline Payment** in OFBiz is simply a convenient, manual process for receiving payments, especially for payment methods that have no automated processing implemented yet. Credit cards will usually have automated payment gateways that OFBiz can talk to for automated payment processing without human intervention. Use "Offline Payment" when encountering say an unknown credit card for which there is no automated payment processing.

Note how the **Quick Ship Entire Order** option now appears in the **Shipment Information** screenlet. Before we ship the ordered item to fulfill the order, we need to be paid.

Shipment Information - 00001 **Ship Group PDF**

Address: Somewhere - A City

Method: UPS Guaranteed Next Day

Update

Splitting Preference: Please wait until the entire order is ready before shipping. **Allow Split**

Gift: This order is not a gift

Quick Ship Entire Order

Pack Shipment For Ship Group [00001]

New Shipment For Ship Group [00001]

View/Edit Delivery Schedule Info

Fulfilling the Order

Since we have received payment in full, it is now time to ship the order. Click on the button that says **Quick Ship Entire Order**.

There are actually a series of actions to be performed in order to ship and fulfill the order. This convenient quick ship function performs those actions with some default values. The order should now be fulfilled.

Invoice Automatically Generated with Payment Applied

OFBiz components and webapps are inter-linked, forming an integrated system that qualifies as an ERP application. Let's take a peek into the webapp "accounting" to see some integration and automation in action. In the **Payment Information** screenlet, in the row **Invoices**, click on the invoice number to view the automatically generated invoice.

Payment Information	
Cash	
Max Amount:	Max Amount: US\$65.30
US\$65.30	[Received]
<hr/>	
Invoices	# 10000 (PDF)

Note how the invoice has a zero **Open** amount, and has **Applied Payments** of **\$65.30**. The payment that we received earlier has been automatically applied to this invoice.

End-to-End Tour Completed

We have now successfully completed an end-to-end tour from making a purchase with the webapp `ecommerce` to processing and fulfilling the sales order with the webapp `ordermgr`. As can be seen, OFBiz already provides quite a comprehensive and usable package for businesses "out of the box", that is without any amendments at all.

Our quick tour brings us through only a tiny fraction of OFBiz. As we go through this book, we will see more possibilities with OFBiz.

Spend some time familiarizing yourself with the back-end management consoles. Don't forget, you are only working on a test system and anything you do can be quickly reverted. In the next chapter we will find out how to revert the changes to the data in the most time efficient way.

Summary

In this chapter, we learned to:

- Obtain and install SVN, the software required to obtain OFBiz
- Use SVN to download OFBiz
- Obtain and install JDK, the tool required to compile and run OFBiz
- Download OFBiz ready to launch
- Create an Eclipse Project
- Compile OFBiz
- Load seed data into OFBiz
- Backup OFBiz data so that we don't need to re-do the lengthy data-loading process in future
- Start OFBiz
- Access OFBiz (URL pattern to construct commands to send to OFBiz)

In the next chapter, we'll be doing some initial tweaks to OFBiz—our first customizations! We will also be learning the structure of OFBiz, so that our customizations can be organized in a sensible way that is compatible with OFBiz conventions.

2

Working with OFBiz

Having taken our first quick tour of OFBiz, let us now take a stab at making small tweaks to OFBiz. This section will give us a brief tour of the behind-the-scenes components in OFBiz.

Specifically, we will be looking at:

- Quick yet powerful customizations, to give us a taste of the power we have with OFBiz.
- Saving and resetting OFBiz data states quickly, to facilitate rapid software development and testing.
- The general structure (physical files and folders) of OFBiz, so we know how to find our way around in OFBiz.

Bear in mind that the customizations described here contain many new terms and concepts. These terms and concepts may not necessarily be explained in detail in this chapter, but later in the book. For now, we will just follow the steps and go along for the ride.

Adding Our First Field

In Chapter 1, we entered some fictitious personal details as an anonymous shopper. OFBiz keeps track of every shopper, even anonymous ones. Let's revisit the "order" webapp to have a look at the anonymous shopper entity that OFBiz created when we bought something via the ecommerce site. Fire an https request, `control/orderview?orderId=WS10000`, to webapp `ordermgr`.

We're looking at the details of the order we created and processed in Chapter 1. In the **Contact Information** screenlet, we see the familiar details of the anonymous shopper who created the order.

The screenshot shows a "Contact Information" screenlet with the following details:

Name	OFBiz Researcher (10000)	New order	Other Orders
Order Notification Email Address	someaddr@somewhere.com (Send a confirmation email)		
Shipping Destination Address	To: Gift Recipient Somewhere A City, AK 123456 USA		
Somewhere - A City		Update	

Looking at the name of the anonymous shopper (**OFBiz Researcher**), we see a system ID (identification number, **10000** in this case) assigned to the shopper. This ID uniquely identifies the shopper in the OFBiz database. Click on the ID to view the detailed profile screen for the shopper. This will bring us to the webapp `partymgr`. We'll be asked to log in again because we just clicked on an external link (external to webapp `ordermgr`).

In the **Contact Information** screenlet here, update the postal address by clicking the **Update** button beside the address.

The screenshot shows a "Contact Information" screenlet with the following details:

Contact Type	Contact Information	Soliciting OK?	Create New
Postal Address	Billing (AP) Address Shipping Destination Address To: Gift Recipient Somewhere A City, AK 123456 United States (Updated: 2008-05-26 18:44:25.75) Request For Bug Fix	0	Update Expire
Create Request			

We'll be brought to the **Edit Contact Information** screen. We will be attempting to change this screen, and the data structure behind it.

Contact Purposes	
Billing (AP) Address (Since:2008-05-26 18:44:25.859)	Delete
Shipping Destination Address (Since:2008-05-26 18:44:25.796)	Delete
Add Purpose	
To Name	Gift Recipient
Attention Name	
Address Line 1	Somewhere
Address Line 2	
City	A City
State/Province	AK
Zip/Postal Code	123456
Country	United States
Allow Solicitation?	<input type="checkbox"/>

Changing the Data

We now change the database entity **PostalAddress**, which is the data structure behind the **Edit Contact Information** screen we want to customize. For now, it is enough to know that the term entity refers to a database table.

Editing the Entity Definition

An "Entity Definition" is an XML element `<entity>` that defines the structure of a data entity (think data structure) as well as its relationships (if any) to other data entities.

In the folder `${OFBizInstallFolder}\applications\party\entitydef`, edit the file `entitymodel.xml`. Go to line 962 where we see:

```
<field name="countryGeoId" type="id"></field>
```

By default Eclipse doesn't show line numbers. To enable this go to: **Window | Preferences... | General | Editors | Text Editors** and select **Show Line Numbers**. Readers can also use *Ctrl+L* to go to a specific line number.

The entity definition for `PostalAddress` should be between lines 949 and 996. The above line should come just beneath the following start tag at line 949:

```
<entity entity-name="PostalAddress"
        package-name="org.ofbiz.party.contact"
        title="Postal Address Entity">
```

and above the end tag at line 996:

```
</entity>
```

Under line 962, insert this:

```
<field name="planet" type="name"></field>
```

We have now added a new field named `planet` to the data entity `PostalAddress`.

Updating the Database

So far, only the entity definition for data entity `PostalAddress` has changed, but the data structure in the actual database hasn't been updated yet. To update the database, simply restart OFBiz.

Restarting OFBiz works because of two configuration settings in the file `${OFBizInstallFolder}\framework\entity\config` file `entityengine.xml`. The settings are on line 140 to 141:

```
check-on-start="true"
add-missing-on-start="true"
```

When OFBiz has completely restarted, we should confirm that the database structure was updated successfully. Open the file `${OFBizInstallFolder}\runtime\logs` and `console.log` and look for the following lines:

```
DatabaseUtil.java:318:WARN ] Entity [PostalAddress] has 18 fields but
table [OFBIZ.POSTAL_ADDRESS] has 17 columns.
DatabaseUtil.java:330:WARN ] Field [planet] of entity [PostalAddress]
is missing its corresponding column [PLANET]
DatabaseUtil.java:1711:INFO ] [addColumn] sql=ALTER TABLE OFBIZ.
POSTAL_ADDRESS ADD PLANET VARCHAR(100)
DatabaseUtil.java:343:INFO ] Added column [PLANET] to table [OFBIZ.
POSTAL_ADDRESS]
```

The following changes can be done while OFBiz is running. Leave OFBiz running from here on.

Changing the Looks

Now that we have changed the data structure, we should change the user-interface to match.

Editing the User-Interface

In the folder \${OFBizInstallFolder}\applications\party\webapp\partymgr\party, edit editcontactmech.ftl. Go to line 32 and remove the extra double-quote just before the > in [<select name="preContactMechTypeId" " >]. We have just fixed our first bug, though this little error doesn't manifest itself since it is tolerated and accommodated by an internet browser such as Firefox.

Insert right above line 185:

```
<#elseif "TELECOM_NUMBER" = mechMap.contactMechTypeId?if_exists>
```

this:

```
<tr>
  <td class="label">Planet</td>
  <td>
    <input type="text" size="30" maxlength="100" name="planet"
           value="${(mechMap.postalAddress.planet) !}">
  </td>
</tr>
```

Checking Our Changes

Start up OFBiz again. Fire an https request to webapp partymgr, which should bring us to the **Find Party** screen. Enter **Last Name** and **First Name**, **Researcher** and **OFBiz** respectively, and click on **Lookup Party**. Our anonymous shopper should turn up. Click on the **Party ID** link (most likely **10000**) to bring up the party details.

Find Party		Lookup Party	Hide Fields
Contact Information :		<input checked="" type="radio"/> None <input type="radio"/> Postal <input type="radio"/> Telecom <input type="radio"/> Other	
Party ID :		<input type="text"/>	
User Login :		<input type="text"/>	
Last Name :		<input type="text" value="Researcher"/>	
First Name :		<input type="text" value="OFBiz"/>	
Party Group Name :		<input type="text"/>	
Role Type :		<input type="text" value="Any Role Type"/> <input type="button" value="▼"/>	
Lookup Party Show all records			

In the **Contact Information** screenlet, edit the postal address by clicking the **Update** button beside the address. We will be brought to the **Edit Contact Information** screen for the anonymous shopper's postal address. Note the new user-interface field we added.

Country	United States
Planet	<input type="text"/>
Allow Solicitation?	<input type="checkbox"/>
Go Back	Save

We have just enhanced OFBiz to serve in an interplanetary environment. Test out the new field by entering say **Jupiter**, and clicking **Save**.

Changing the Flow

If we imagine that the database or persistence layer of OFBiz is like the engine of a car, the user-interface layer would be the steering wheel (or gas pedal or any other interfaces with the human user), and the flow layer would be the wiring between the two. Working the flow allows us to determine whether the gas pedal activates the engines or the windshield wipers.

Leave OFBiz running for this exercise. Do not shut it down.

Let us now imagine that all postal packages on Mars have problems of being too light to withstand the strong winds there. We will need to issue an advisory to anyone dealing with Martian addresses.

Rewiring the "Save" (Update Postal Address) Button

We will rewire the **Save** button to point to a new screen upon a successful update of the postal address.

In the folder \${OFBizInstallFolder}\applications\party\webapp\partymgr\WEB-INF, edit the file controller.xml. Replace line 131:

```
<response name="success" type="view" value="editcontactmech"/>
```

with:

```
<response name="success" type="view" value="PostalAddressAdvisory"/>
```

We've just rewired the **Save** button to go to a view map called **PostalAddressAdvisory**. A view map is like a view or screen that is presented to the end-user. View maps are covered in detail in Chapter 8.

We need to extend the original `controller.xml` file. We create a new file called `extended.xml` right beside `controller.xml`. The idea is to minimize changes to `controller.xml` and cleanly separate new additions to an organized consolidated area (`extended.xml` in this case). We now create a view map called **PostalAddressAdvisory**. Enter this into `extended.xml`:

```
<?xml version="1.0" encoding="UTF-8" ?>
<site-conf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/
        site-conf.xsd">
    <view-map name="PostalAddressAdvisory" type="screen"
        page="component://party/widget/partymgr/
            OurPartyScreens.xml#PostalAddressAdvisory"/>
</site-conf>
```

To include `extended.xml` in `controller.xml`, we use the `<include>` element in `controller.xml`. Right above line 23:

```
<description>Party Manager Module Site Configuration File</
description>
```

insert this:

```
<include location="component://party/webapp/partymgr/
    WEB-INF/extended.xml"/>
```

Creating the New Widget Screen

Having created the view map that points to a file `OurPartyScreens.xml`, and specifically a screen widget (explained in Chapter 3) called **PostalAddressAdvisory**, we now have to create that non-existent screen widget.

In this folder `${OFBizInstallFolder}\applications\party\widget\partymgr`, create a file `OurPartyScreens.xml`. Enter into `OurPartyScreens.xml` this:

```
<?xml version="1.0" encoding="UTF-8"?>
<screens xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/
        widget-screen.xsd">
    <screen name="PostalAddressAdvisory">
        <section>

        <actions>
            <set field="titleProperty" value="PageTitleEditContactMech"/>
            <set field="headerItem" value="find"/>
```

```
<set field="tabButtonItem" value="editcontactmech"/>
<set field="labelTitleProperty"
      value="PageTitleEditContactMech"/>
<script location="component://party/webapp/partymgr/
WEB-INF/actions/party/editcontactmech.bsh"/>
</actions>
<widgets>
  <decorator-screen name="main-decorator"
    location="${parameters.mainDecoratorLocation}">
    <decorator-section name="body">
      <section>
        <widgets>
          <platform-specific><html>
            <html-template location="component://party/webapp/
              partymgr/party/
              postaladdressadvisory.ftl"/>
          </html></platform-specific>
        </widgets>
      </section>
    </decorator-section>
  </decorator-screen>
</widgets>
</section>
</screen>
</screens>
```

Note how the `<html-template>` element above points to a file `postaladdressadvisory.ftl`. We'll need to create that file next.

Creating the FTL File

In the directory `${OFBizInstallFolder}\applications\party\webapp\partymgr\party`, create a new file named `postaladdressadvisory.ftl`. In this file, enter the following lines:

```
<h2><u>Bogus Postal Address Advisory</u></h2>

<br/>
<div style="width:20%">
Please be advised that Martian gravity is just one third of Earth's,
and winds can be as strong as 400km/h. All postal packages to Mars
must have a mass of at least 20kg to facilitate safe handling.
</div>

<br/>
<div style="width:20%">
Please be advised also that this advisory is completely bogus, and
that you
```

```
should ignore it completely.  
</div>  
  
<br/>  
  
    <a href="<@ofbizUrl>editcontactmech?partyId=\$ {partyId}&contactMechId=  
    \${contactMechId}</@ofbizUrl>">Go Back To Editing Postal Address</a>
```

Checking Our New Wiring

Go back to the **Edit Contact Information** screen described earlier in this chapter. We can do that via the **Find Party** screen in webapp `partymgr`, as described in the section called *Checking Our Changes*.

Enter for field **planet** a value, say **Venus**, or any other value at all. Note how we will now be brought to the advisory screen we just created. Click on **Go Back To Editing Postal Address** to go back to the **Edit Contact Information** screen.



We have successfully changed the flow of things by splicing in an advisory screen.

More to the Flow

Our advisory is regarding Martian postal addresses. What if the postal address we're updating is not Martian? We need to make our wiring smarter. We want to add some preprocessing before showing the advisory.

Creating the Wiring for Preprocessing

In the folder `${OFBizInstallFolder}\applications\party\webapp\partymgr\WEB-INF`, edit `extended.xml`. Insert above line 4:

```
<view-map name="PostalAddressAdvisory" type="screen"
    page="component://party/widget/partymgr/
        OurPartyScreens.xml#PostalAddressAdvisory"/>
```

this:

```
<handler name="bsf" type="request"
    class="org.ofbiz.webapp.event.BsfEventHandler"/>
<request-map uri="PostalAddressAdvisory">
    <security https="true" auth="true"/>
    <event type="bsf"
        invoke="org/ofbiz/party/party/postalAddressAdvisory.bsh"/>
    <response name="isMars" type="view" value="PostalAddressAdvisory"/>
    <response name="notMars" type="view" value="editcontactmech"/>
</request-map>
```

Note the `<event>` element's attribute `type` has a value of `bsf`. It stands for **BeanShell Framework**. That means we are coding with the BeanShell scripting language in the file `postalAddressAdvisory.bsh`. Although quick to prototype, and ideal for the sake of a quick example, BeanShell is slower to execute than compiled java, is very difficult to debug, and at the time of writing is in the process of being replaced in OFBiz by Groovy.

Next, we point the **Save** button (on the **Edit Contact Information** screen) to this new wiring for preprocessing. Replace line 131 in `controller.xml`:

```
<response name="success" type="view" value="PostalAddressAdvisory"/>
```

with:

```
<response name="success" type="request"
    value="PostalAddressAdvisory"/>
```

That line no longer points to a view map but to the request map we had just created in `extended.xml`. Request maps are discussed in detail in Chapter 8.

Creating the Preprocessing

Note how the element `<event>` in our request map above points to a file `postalAddressAdvisory.bsh`. That is the file that we must create, the file that contains the pre-processing we need.

In folder `${OFBizInstallFolder}\applications\party\script\org\ofbiz\party\party`, create a new file `postalAddressAdvisory.bsh` and enter into it the following:

```
import org.ofbiz.party.contact.*;
String partyId = request.getParameter("partyId");
```

```
Map mechMap = new HashMap();
ContactMechWorker.getContactMechAndRelated(request, partyId, mechMap);

Map postalAddress = (Map)mechMap.get("postalAddress");
if (postalAddress == null) return "notMars";
String planet = (String)postalAddress.get("planet");

if (planet == null || !planet.equalsIgnoreCase("Mars"))
    return "notMars";

return "isMars";
```

Testing Our Smarter Wiring

Go to the **Edit Contact Information** screen for our anonymous shopper. Enter any value for the field **planet**, say **Pluto**. Notice how the flow goes straight back to the **Edit Contact Information** screen, skipping our advisory.

Now enter a value of **Mars** for the field **planet**. Our advisory now shows up.

If OFBiz were a car, it's like we had rewired our gear box to activate a warning whenever we shift into reverse-gear. A warning that says: "Make Way! Backing Up!". Moving the gear stick still shifts the gears, as usual. We had just spliced in an additional workflow, without interrupting the original processes.

And that's our first taste of tweaking this immense beast that is OFBiz.

Some Changes Possible with Engines Running

From the previous sections, we noticed that only the data entity changes require an engine shutdown and a subsequent restart. Changing the looks and the flow of OFBiz can be done on-the-fly.

This is one of the many key advantages that OFBiz offers—the ability to develop the software rapidly. In contrast, anyone familiar with Java (the programming language) will know that its clean object-oriented structure and comprehensive libraries are offset by its traditional "change, re-compile, test" cycle that is common to many programming languages (such as C/C++). Between changing the software code and re-compiling, there is usually a need to shut down the software. Shutting down and then restarting large server-based software can take time, possibly several minutes, much longer than the few seconds of downtime that programmers hope for in-between the tens of thousands of changes they make to a typical software program.

OFBiz provides many development methods that don't require a re-compile for every change made. In general, all user-interface changes in OFBiz do not require OFBiz to be shutdown prior to those changes. As we had seen, the user-interfaces can be changed and tested without a re-compile. Changing the flow of OFBiz also doesn't require the engines to be first stopped.

A few types of changes do require an OFBiz shutdown. Changes to the data entity are one. Changes to OFBiz configuration files are another, though effort is underway to remove that constraint. Changes to Java code will definitely require OFBiz to be shutdown prior to a re-compile.

For good measure, let us now make another change to the user-interface while OFBiz is still running. The field **Planet** should come right after **Zip/Postal Code**, and before **Country**. A single country or nation could span several planets.

In the folder \${OFBizInstallFolder}\applications\party\webapp\partymgr\party, edit editcontactmech.ftl. Cut out the chunk from line 185 to 190 that is the HTML element <tr>:

```
<tr>
    <td class="label">Planet</td>
    <td>
        <input type="text" size="30" maxlength="100" name="planet"
               value="${(mechMap.postalAddress.planet) !""}">
    </td>
</tr>
```

and paste it immediately above line 166:

```
<tr>
    <td class="label">${uiLabelMap.CommonCountry}</td>
</tr>
```

Go to the **Edit Contact Information** screen for our anonymous shopper to see this change immediately take effect.

Resetting Our Play Area Quickly

Before we move on to learn the OFBiz framework proper, we need to get some basic kung fu that will tremendously aid our future exploration of OFBiz. This section describes some common short-cuts or "rapid development techniques" that will convert us from mere tinkerers into serious scientists.

Skipping Some Pre-Ignition Processes

In Chapter 1 we learned how to backup the Derby data files. In this section, we will look at how we can use that backup.

Imagine now that we want to show Mr. Evaluator, a noted ERP systems evaluator, our OFBiz. We want it to be pristine and clean, without the nonsensical test data we had entered thus far. We will first need to destroy the existing data. We delete the existing Derby data files. The Derby data files are in `${OFBizInstallFolder}\runtime\data\derby`, contained in a folder **ofbiz** there. Delete the folder **ofbiz** there.

Restoring Derby Data Files

We need to unzip our backup of the Derby data files which we created in Chapter 1. You did backup the Derby data files then, right? If you didn't, you will have to go through the long data-loading process again, possibly 15 minutes or more. See Chapter 1 for that process again, and then backup the Derby data files right after.

Unzip our backup of the Derby data files into the folder `${OFBizInstallFolder}\runtime\data\derby` files. That's it. We now have a pristine, clean set of data for OFBiz, and Mr. Evaluator won't have to see the weird test data we keyed in.

Removing the Web Server (Catalina) Work Files

We now need to remove the web server work files. Go to folder `${OFBizInstallFolder}\runtime\catalina` and delete the folder `work`.

Work files contain session information from previous visitors as the database references to these visits no longer exist in the database, OFBiz will complain.

Updating the Database with Our Data Entity Changes

In the section called *Changing the Data*, we changed the data entity `PostalAddress`. We subsequently changed some user-interfaces to match. Since we now only have the stock plain-vanilla OFBiz database, we will need to update the database again. This process is exactly the same as the section called *Updating the Database*. Restart OFBiz to update the database.

Showing Off Our Spanking New OFBiz Installation

The restoration of the Derby data files and the deletion of web server work files should take no more than a few minutes. Our installation of OFBiz is now pristine again, untouched by any prior usage.

It's time to show off our spanking new OFBiz installation to Mr. Evaluator. We buy four round gizmos as an anonymous shopper, key in details for the anonymous shopper, and then confirm and create the order. We log in to webapp `ordermgr` and pull up the order's details, and approve the order. We are about to receive some offline payment. But wait! Mr. Evaluator has something to say!

Tripping Up Our Plan

At this point, Mr. Evaluator suddenly asks a question that would derail the flow of our presentation plan. Or would it? "What if I ship the order before receiving payment?" Mr. Evaluator asks.

It would seem we now have only one option; to tell Mr. Evaluator to let us finish our presentation first and then go through other "scenarios" from square one. Fortunately, we can start from square one easily, after a few minutes of restoring the original Derby data files and clearing the web server work files.

Or we could do something more impressive here, and say the following:

- "computer, freeze program, and store save-point"
- "computer, run scenario B that Mr. Evaluator just described"
- "computer, run scenario A from last save point"

Well, OFBiz still doesn't take voice commands. What we're about to do is exactly as impressive as the above, though without the voice commands.

Storing a Save-Point to Dramatically Ease Testing

Before we can store a save-point, we need to shut down OFBiz first. Do so now!

Archiving Derby Data Files and Web Server Work Files

The process to backup the Derby data files was already described, but the process to backup the web server work files is yet to be covered. We'll cover both processes here so we can have it all in one place.

Backing Up the Derby Data Files

The Derby data files are in `${OFBizInstallFolder}\runtime\data\derby`, contained in a folder named `ofbiz`.

Go to `${OFBizInstallFolder}\runtime\data\derby` and zip up the Derby data files into a single archive say `ofbiz_4_0_r589272_SavepointA.zip`. Keep the archive safe somewhere.

Backing Up the Web Server Work Files

The web server work files contain web sessions and session IDs. OFBiz stores some web visit details in its database, and these details tie in with the web server work files. Hence, the web server work files need to be correctly matched with the Derby data files. That means we save both and archive them together when creating a save-point.

The web server work files are in the folder `${OFBizInstallFolder}\runtime\catalina`, contained in the folder `work`. Backup the folder `work` and keep the archive safe somewhere, together with the corresponding Derby data files backup we created for this save-point, say in an archive `ofbiz_4_0_r589272_SavepointA_web.zip`.

Computer, Run Scenario B that Mr. Evaluator just Described

We start up OFBiz, and go back to webapp `ordermgr` to view our order. This time, instead of receiving payment first, we do a quick ship first. We then receive an offline payment for the whole order (\$65.30).

Following the tour outlined previously, we click on the automatically generated invoice for the order. And we see a difference! The invoice's "Open" amount is still \$65.30, while "Applied Payments" is \$0.

Mr. Evaluator lets out a gasp of surprise, and asks why received payments are not automatically applied to automatically generated invoices. We make a quick save and say that OFBiz is really a huge beast that has many work-in-progress areas. To solidify the save, we add that "this auto-application of received payments is already implemented, though it is hooked up somewhere else in OFBiz".

We now move quickly to show Mr. Evaluator that OFBiz does automatically apply received payments.

Restoring a Save-Point

Before we can restore a save-point, we need to shut down OFBiz first. Do so now. Since we are discarding the current state of OFBiz, we must first remove the existing Derby data files and web server work files. Do so now!

Restoring the Derby Data Files

Unzip into \${OFBizInstallFolder}\runtime\data\derby the Derby data files archive `ofbiz_4_0_r589272_SavepointA.zip`, which we had created when we stored the save-point we are now restoring. We should have restored the folder `ofbiz`.

Restoring the Web Server Work Files

Unzip into \${OFBizInstallFolder}\runtime\catalina the web server work files archive `ofbiz_4_0_r589272_SavepointA_web.zip`. We should have restored the folder `work`.

Computer, Run Scenario A from Last Save-Point

We are now back to looking at an order that has neither been shipped nor received payment. We now receive payment first, and then perform a quick ship. We look at the automatically generated invoice, and show Mr. Evaluator that the received payment was automatically applied to the invoice.

We wait for applause. None comes. Mr. Evaluator then asks, "how long will it take you to put this logic, this auto-application of received payment, into scenario B?". Don't sweat! That's what this book is for. We'll learn how to perform such tweaks and customizations and enhancements. So let's get learning!

The Structure of OFBiz in General

Let's take a quick overview of the general structure of OFBiz, so we know how to move around, and know where to find what. Note that the following describes conventions in OFBiz. We can certainly break those conventions and misplace files, but that will make things more difficult to maintain and understand. For example, a postal address is always structured to be ordered in descending precision of geography, with the more pinpoint address (house number, street name) at the top and the more general locale (state, province, country) at the bottom. Postal services will find it easier to deliver our mail if we follow that convention. Similarly, an organized OFBiz structure makes OFBiz easier to work with.

Components in OFBiz

OFBiz files are organized according to components, with each component being contained in a single folder.

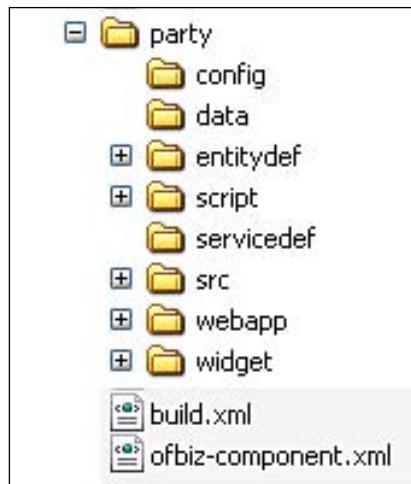
The core application OFBiz components can be found in `${OFBizInstallFolder}\applications`, where we have already seen the "ecommerce" component (folder `ecommerce`) and the "party" component (folder `party`). To qualify as a core component, a component must be integrated into OFBiz and contain functions that serve a wide denomination of users. Such components are so commonly used that they have become an integral part of OFBiz.

The framework OFBiz components can be found in `${OFBizInstallFolder}\framework`. These components are the foundations of OFBiz. They are responsible for database access, caching, rendering the screens, managing transactions, and many more low-level tasks used by the application components. Very often, it is not necessary to understand the code found inside these components – only how to use the features is important.

Special purpose components are those that don't quite qualify as core OFBiz components. They serve only a small fraction of OFBiz users. OFBiz can (or should) function without these components, since they are (or should be) optional. Such components reside in `${OFBizInstallFolder}\specialpurpose`.

Each OFBiz component is pretty much self-contained (other than for its relationships with other components). Each OFBiz component has data entity definitions, view definitions, flow definitions (concepts touched on tangentially in the section called "Doing Our First Customization"), the whole works. Yet, OFBiz components often need to work with each other.

The typical OFBiz component structure may consist of configuration files, seed data definition, entity definition, flow logic, service definition (Service Engine discussed later in book), Java code, webapps, and view definitions. All of these will be discussed in greater detail in later chapters.



Referencing Components in OFBiz

In OFBiz, components are referenced via special URIs (Universal Request Identifier) that we call "component URIs". A component URI is of the format `component : //<component-name>/relative-path`, where `<component-name>` is the name of any component in OFBiz.

The name of a component is found in the definition of the component, that definition residing in the file `ofbiz-component.xml`, a file that we will explore in the section called *Creating Our Own OFBiz Component*. Each OFBiz component has its own `ofbiz-component.xml`.

To see a list of all OFBiz components and their names, go to the **View Components** screen of webapp `webtools` at <https://request.ViewComponents>. The second column of that list will show the root folder of each component.

In this book (not in OFBiz code examples), we will from now on refer to these components root folders with this format: `${component : <component-name>}` . For example, the folder `${OFBizInstallFolder} \applications\party` will be referred to as `${component : party}` .

Creating Our Own OFBiz Component

Having learned about an OFBiz component's file structure convention, let us now create our very first OFBiz component. We will call this component **learning** and will use it to perform our messy mad-scientist work and experimentation, far away from the official and sensitive parts of OFBiz.

Creating the Component

Custom OFBiz components are usually created in the folder `${OFBizInstallFolder}\hot-deploy`. So long as the structure criteria is met, components placed into hot-deploy will be automatically loaded after those in the framework and applications. There is no need to specify them in `component-load.xml` files, as has to be done in the framework and applications. So, in `hot-deploy`, create a new folder `learning`. This can be done very quickly from our Eclipse project's **Navigator** view by right-clicking on the `hot-deploy` folder, selecting **New | Folder**, and entering its name.

In the folder `learning` we just created, create a new file named `ofbiz-component.xml`. Again, this can be done in Eclipse by right-clicking on the `learning` folder and selecting **New | File**. In this file, enter the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="learning"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/
        ofbiz-component.xsd">
</ofbiz-component>
```

Note that the OFBiz component name we have chosen is **learning** (see attribute `name` of element `<ofbiz-component>`). This name must be unique across the entire OFBiz implementation. To see a list of components and their names and other details, fire an `https` request `ViewComponents` to `webapp webtools`.

Using Our Component

Here, we take the opportunity to clean up the mess we created in the "party" component. Along the way, we will learn to use the OFBiz component we just created. We will also learn some neat OFBiz mechanisms for "clean extensions". "Clean extensions" is code that extends original OFBiz code while minimizing changes to the original OFBiz code.

Cleaning Up Our Mess in the "party" Component

There are a total of 4 files we need to move out of the "party" component. These files are the ones we created and which don't belong to the "party" component:

- `extended.xml`
(In folder `${component:party}\webapp\partymgr\WEB-INF`)
- `OurPartyScreens.xml`
(In folder `${component:party}\widget\partymgr`)
- `PostalAddressAdvisory.ftl`
(In folder `${component:party}\webapp\partymgr\party`)
- `PostalAddressAdvisory.bsh`
(In folder `${component:party}\script\org\ofbiz\party\party`)

We will move those files into folder `${component:learning}`.

In `${component:learning}`, create three nested folders `webapp\partymgr\WEB-INF`. Move the file `extended.xml` into the deepest folder. In the folder `${component:party}\webapp\partymgr\WEB-INF`, edit the file `controller.xml` and change line 23:

```
<include location="component://party/webapp/partymgr/  
WEB-INF/extended.xml"/>
```

to:

```
<include location="component://learning/webapp/partymgr/  
WEB-INF/extended.xml"/>
```

Note how the new component URI now points to the component `learning`, instead of the component `party`.

In `${component:learning}`, create two nested folders `widget\partymgr`. Move the file `OurPartyScreens.xml` into the deepest folder. Edit `extended.xml` and change line 13:

```
<view-map name="PostalAddressAdvisory" type="screen"  
page="component://party/widget/partymgr/  
OurPartyScreens.xml#PostalAddressAdvisory"/>
```

to:

```
<view-map name="PostalAddressAdvisory" type="screen"  
page="component://learning/widget/partymgr/  
OurPartyScreens.xml#PostalAddressAdvisory"/>
```

In the folder \${component:learning}\webapp\partymgr, create a folder party. Move the file PostalAddressAdvisory.ftl into that folder. Edit OurPartyScreens.xml and change line 20:

```
<html-template location="component://party/webapp/partymgr/party/  
PostalAddressAdvisory.ftl"/>
```

to:

```
<html-template location="component://learning/webapp/partymgr/party/  
PostalAddressAdvisory.ftl"/>
```

Converting the BeanShell to a Java Event

Finally we come to the BeanShell file. Rather than simply copying this over, now would be an ideal time to take a look at the far more common approach to dealing with events in OFBiz: Java.

Edit extended.xml and replace line 8:

```
<event type="bsf"  
invoke="org/ofbiz/party/party/postalAddressAdvisory.bsh"/>
```

with:

```
<event type="java" path=" org.ofbiz.learning.learning.learning"  
invoke="postalAddressAdvisory"/>
```

We also need to tell the request-map that the type of event has changed, so in the same file replace:

```
<event type="bsf"  
invoke="org/ofbiz/party/party/postalAddressAdvisory.bsh"/>  
<response name="isMars" type="view" value="PostalAddressAdvisory"/>
```

with:

```
<event type="java"  
path="org.ofbiz.learning.learning.LearningEvents"  
invoke="postalAddressAdvisory.bsh"/>
```

In the folder \${component:learning}, create five nested folders src\org\ofbiz\learning\learning.

We are going to insert a Java class into this folder but first we have to tell Eclipse that this folder is going to contain Java code. To do this right-click on the blue, project root, **ofbiz4.0** folder from the **Navigator** box and select **Properties**, then **Java Build Path**. Select the **Source** tab and click **Add Folder....** In fact we want Eclipse to know that each file in the whole `src` folder is going to be Java class, so navigate to this folder, highlight the checkbox and click **OK** to add this folder as a **source folder** and **OK** again to leave the **Project Properties** dialog box and return to our project. Now, if we right-click on the `party` folder and choose to add a new class, the correct package information will be added automatically to the class. Let's call our class `LearningEvents`. Just adding `LearningEvents` in the **Name** field and clicking **Finish** is enough. There is no need to add `.java` to the end of the name; Eclipse will do this for us.

In the new class, we want to add just one method called `postalAddressAdvisory` which is invoked from the controller entry we just changed. The final class should look like this:

```
package org.ofbiz.learning.learning;

import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.ofbiz.party.contact.ContactMechWorker;

public class LearningEvents {

    public static String postalAddressAdvisory(HttpServletRequest
        request, HttpServletResponse response){

        String partyId = request.getParameter("partyId");
        Map mechMap = new HashMap();
        ContactMechWorker.getContactMechAndRelated(request, partyId,
            mechMap);

        Map postalAddress = (Map)mechMap.get("postalAddress");
        if (postalAddress == null) return "notMars";

        String planet = (String)postalAddress.get("planet");
        if (planet == null || !planet.equalsIgnoreCase("Mars"))
            return "notMars";

        return "isMars";
    }
}
```

The method must be made static as the OFBiz framework calls the method and does not instantiate the class `LearningEvents`. The framework also takes care of passing in the parameters `HttpServletRequest request` and `HttpServletResponse response`.

Unfortunately, adding the learning folder under `hot-deploy`, does not automatically include it in the project compilation. First of all, a `build.xml` file must be created under the learning folder at the same level as the `ofbiz-component.xml` file. A quick trick here is to copy the `build.xml` file from `${OFBizInstallFolder} \ applications\party` to `${OFBizInstallFolder} \ hot-deploy\learning` and replacing everywhere it says **Party** to **Learning**—there are four places on lines 21, 29, 30, and 112. We now need to change just one dependency. Since the party component code depends on some code in the content component, and our module depends instead on some code in the party module, we must change line 51 from:

```
<fileset dir=".../content/build/lib" includes="*.jar"/>
```

to:

```
<fileset dir=".../applications/party/build/lib" includes="*.jar"/>
```

Once this has been done, make sure OFBiz is not running and in the **Ant** window on the right-hand side, double-click the **OFBiz Main Build** entry to compile and build the whole project. Once compiled, select the learning folder in the **Navigator** view and hit *F5* to refresh the project, you will now see a new folder called `build` which contains the classes and lib folders. Within the `lib` folder there should be a file called `ofbiz-learning.jar`.

In some of the future examples, there may not always be a need to recompile the entire project. By dragging the `build.xml` file from your learning folder over into the **Ant** window, you can quickly double-click this while you are working on something in the main window. This can cut the compilation time down from a minute or more to a second or less.

Finally, before we restart, we must add the folder that has just been created to the classpath. To do this, open the rather empty `ofbiz-component.xml` file and insert:

```
<classpath type="jar" location="build/lib/*"/>
```

Rebuild the whole project and restart.

Clean Extension Strategies Employed

A few clean extension strategies were employed in the above.

The most obvious strategy we can see above is inter-component referencing via the component URIs. Being able to reference another component, say referencing the **learning** component from the **party** component in our case above, allows us to keep components largely unchanged while extending them with a separate component. In our examples above, the **party** component was only minimally changed merely to point to the actual extensions coded in the **learning** component.

Another trick is the "controller include" strategy with the `<include>` element, which lets us extend a `controller.xml` file. As with the inter-component referencing strategy described above, this trick lets us keep the `controller.xml` file clean with a mere pointer to the extensions coded in the file `extended.xml`.

By employing clean extension strategies from the very beginning we are able to cleanly separate the core code from our own bespoke code. When it comes time to upgrade, this will considerably cut down the time and complexity of this task. It could even mean the difference of being able to upgrade or not.

Checking that Our Move was Successful

Restart OFBiz, go to the webapp **party** and bring up any postal address for edit. Ensure that the original behavior with the field **planet** is still retained. If so, our move was successful, and the **party** component is now free of our new files.

A Bit More Mess Remains

The file `${component:party}\entitydef\entitymodel.xml` was changed to add a new field **planet**.

The file `editcontactmech.ftl` in the folder `${component:party}\webapp\partymgr\party` still contains a chunk of enhancement code to display the new field **planet**. Clean extension tricks in this area will be described in the next chapter on **screen widgets**.

Webapps in OFBiz

An OFBiz component by itself cannot be accessed by end-users. It is simply a means of organizing OFBiz into individual parcels focused on dealing with individual aspects of ERP software. Webapps or web applications provide the front-end through which end-users can work with and use OFBiz.

Think of a webapp as an individual "office opened for service". Such a virtual "office" can recognize and respond to several different "requests" (listed as request maps in `controller.xml`).

Webapps are contained inside OFBiz components. Typically, each OFBiz component has one webapp, but can actually have more. The **ecommerce** component has two, named **ecommerce** and **e-comclone**, because OFBiz developers wanted to showcase how multiple storefronts can be set up based on the single consistent set of functionalities coded in the **ecommerce** component. The **product** component also has two, named **catalog** and **facility**, because product management ("catalog") and inventory handling ("facility") work off of logic that are closely intertwined.

As mentioned before, OFBiz components are simply a way to organize OFBiz into manageable chunks or logically distinct aspects of ERP software. Technically, it is certainly possible to put all 17 webapps in OFBiz under a single component, though to do so would be quite illogical and would make things very unwieldy.

Webapps are possible in OFBiz because it has an embedded web server called Tomcat.

Creating Our First Webapp

All that makes a webapp is a `<webapp>` element in an OFBiz component's `ofbiz-component.xml` file. Edit `${component:learning}\ofbiz-component.xml` and insert into the `<ofbiz-component>` element a `<webapp>` element like this:

```
<webapp name="learning"
        title="Learning"
        server="default-server"
        location="webapp/learning"
        base-permission="NONE"
        mount-point="/learning"/>
```

From this fact, we will know instantly that the folder `${component:learning}\webapp\partymgr` is not a webapp; it is not defined in `ofbiz-component.xml` as a webapp location. Instead the location is pointing to `webapp\learning`. This path does not yet exist in our project and must be created before we restart.

The `title` attribute determines the text shown in the `appbar`, which is the toolbar that lists all webapps in OFBiz. The `appbar` is right below the OFBiz logo that says **The Apache Open for Business Project** and is sorted in alphabetical order. If an attribute `app-bar-display` is present and has a value of `false`, the webapp will not appear in the `appbar`.



The `location` attribute is a path to our webapp folder, relative to the component folder. In this case, the absolute path to our webapp folder is `${component:learning}\webapp\learning`, so the relative path to it (after chopping off the `${component:learning}` portion) is `webapp\learning`.

The `mount-point` attribute determines what URL the end-user will use to access our webapp. In this case, our end-user will access our webapp with a URL like `http://localhost:8080/learning`. We can just as easily mount it somewhere else fanciful, like at "special-learning", but let's not make it harder for us to type out URLs.

The `name` attribute doesn't seem to be important, as webapps are never referenced in OFBiz by their names. Still, this may change in the future, and there might be webapp URIs similar to component URIs. For now, we should try to keep the `name` attribute exactly the same as the `mount-point` attribute.

The other attributes will be explained in more detail later in the book.

Webapp URIs in this Book

Although webapp URIs do not yet exist in OFBiz, we will use a short hand term to refer to webapp roots in this book, for the sake of brevity and readability. The short-hand will be of this format: `${webapp:<webapp-name>}`. Therefore, a pathname say `${component:learning}\webapp\learning` will be referred to as `${webapp:learning}`. For paths that are mere folders and not webapps, like `${component:learning}\webapp\partymgr`, they will continue to be referred to via that form.

Webapp names will be derived from the `mount-point` attribute minus the leading "/", not the `name` attribute. Mount points need to be unique. It makes no sense to have name different from `mount point`. Also it makes no sense to mount two webapps with different name attributes onto the same `mount point`.

Testing Our First Webapp

We want to know if our webapp learning has been set up correctly and is accessible to the end-user.

In the folder \${webapp:learning}, create a new file called index.html and enter into it the following:

```
This is webapp "learning" responding with this text.<br/>
If you're seeing this, you have successfully called on the
webapp.<br/>
Do not call SETI; this is not a message from "out there".
```

Restart OFBiz and fire an http request index.html to the webapp learning. You should see that silly message we just entered into the file.

The Model-View-Controller Architectural Pattern

OFBiz uses a tried-and-tested software organizational structure called the **Model-View-Controller (MVC)**. Since this book is organized into parts following the MVC architecture, we will take a quick overview of how OFBiz files correspond to MVC.

The MVC in Plain English

The Model refers to the data structure of the software. Data structures are often used to virtually describe a real-world system, much like how a model railway system with model trains describe the real railway system in the real world. A typical "model" may consist of a "customer" entity and a "sales person" entity in a "many-to-one" relationship that mimics a real-world situation where one sales person serves several customers. As can already be seen here, a "model" is a collection of data entities in a network of relationships with one another.

The View refers to the "outsides" of the software, the component that is handled by the human user (end-user) – the user-interface.

The Controller refers to the wiring between the Model and the View. The Controller part can be huge, and is almost always larger than the Model and the View. The Model and View are quite static. The Model being a fixed collection of data entities in a fixed network of relationships, and the View being a collection of user-interfaces that may be described as static dashboards. In contrast, the Controller is the dynamic behavior defined by nothing less than a hairball of wiring.

This section describes the MVC architecture of OFBiz in the context of an OFBiz component. That is, all file locations described here are relative to the root of an OFBiz component. As mentioned in the previous section, OFBiz components are self-contained save for their working relations with each other. So, each component has its own MVC structure.

The Model in OFBiz

The Model in OFBiz is defined in data definition XML files, such as in the file folder `${component:party}\entitydef entitymodel.xml` which we played with earlier in the chapter in *Changing the Data*.

Such files are always (or should always be) located in folders named `entitydef`. Each OFBiz component can have one (or zero) of such folders. It follows that each OFBiz defines its own "little world" or "distinct model". The **party** component, for example, has a model that describes parties or people, such as shoppers and back office staff. And that "distinct model" is cleanly separated from the model in say **order** component where customer orders are modeled.



Within each "distinct model", there can be a network of relationships between individual data entities. This is seen in the way the entity *Party* is in some way linked to entity *PostalAddress* so that people can have postal addresses attached to them.

Between two separate "distinct models", there can be relationships too. For example, the entity *Party* in the **party** component can be linked to the entity *Order* in the **order** component, such that shoppers can be linked to their orders.

The View in OFBiz

Views or user-interfaces in OFBiz can reside in two different places, depending on which technology is being used to define those views. The technologies themselves will be discussed in more detail later on in the book.

When using the OFBiz widget technology, user-interface definitions reside in folders named `widget`. Each OFBiz component can have one such folder. Here, it must be noted that OFBiz does have some of these files misplaced in `webapp` folders. OFBiz is very large and still has many work-in-progress areas. OFBiz may also contain some remnants of former programmer habits and mistakes or old fashioned techniques, or perhaps even spelling mistakes, having been unnoticed in development for six years.

When using HTML or FTL technologies, user-interface definitions should reside in folders named `webapp`. Note that this excludes the folder `WEB-INF` inside each `webapp` folder – that folder contains **flow** components or **business logic**.



The Controller in OFBiz

Controller components or "flow" or business logic in OFBiz can reside in three different places, depending on the technology used to define business logic.

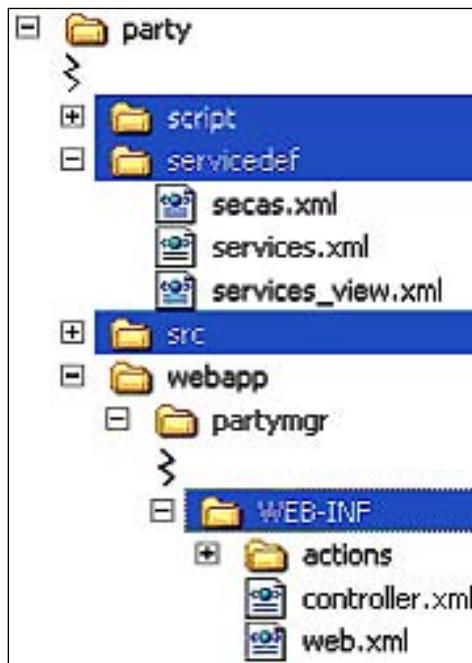
When using pure Java, business logic definitions should reside in folders named `src`.

When using Minilang or BeanShell, business logic definitions usually reside in folders named `script`, and also in the folder `WEB-INF\actions` inside `webapp`.

When using **Event-Condition-Action (ECA)**, business logic is defined in the folder `servicedef`, in the file `secsas.xml` when using Service ECA; and in the folder `entitydef` in the file `eecas.xml` when using Entity ECA.

When using OFBiz Services, business logic definitions span over the folders `servicedef` and `script`. In the folder `servicedef`, files with names starting "services" (for example `services.xml`, `service_view.xml`) declare the usage pattern of services. Complementing that, in the folder `script` or `src`, we have the actual implementation or definition of those services.

The files `controller.xml` and `web.xml` in the folder `WEB-INF` are also considered controller components in the MVC architecture.



Other Files in an OFBiz Component

There are other files besides those described above, files that don't fit into the MVC architecture. Some are configuration files; some are seed data files. These files will be explained when we need to deal with them later in this book.

Summary

In this chapter, we looked at:

- Some quick yet powerful tweaks to OFBiz, covering the whole spectrum of MVC.
- Saving data states by archiving.
- Derby data files in the folder `${OFBizInstallFolder}\runtime\data\derby\ofbiz`.
- Catalina work files in the folder `${OFBizInstallFolder}\runtime\catalina\work`.
- Restoring data states by restoring the above archives.
- The structure of OFBiz, which is made up of components, with zero or more webapps in each component.
- Creating a component of our own with an `<ofbiz-component>` element in the file `ofbiz-component.xml`.
- Creating a webapp of our own with a `<webapp>` element within a `<ofbiz-component>` element.
- Component URIs of the form `component://<component-name>/<relative-path>`
- This book's component paths of the form `${component:<component-name>}\<relative-path>`.
- This book's webapp paths of the form `${webapp:<webapp-name>}\<relative-path>`.
- The MVC architecture in brief, and how OFBiz uses it.
- The folder structure of OFBiz, so we know how to get around in OFBiz.

If the tweaks in this chapter look complex, that's just because OFBiz is so full of possibilities. We next plunge into the OFBiz framework proper, so we can learn to do non-trivial tweaks and more. Before long, we should be able to do a complete and competent customization of OFBiz for just about any business we fancy.

So, let us buckle up and delve into the OFBiz framework, and be "Open For Business" in no time! In the next chapter, we will start with a screen widget which is, a part of the "View" portion of the MVC.

3

Screen Widgets

Screen widgets are part of the OFBiz Widget toolkit. They are the front line of the View element of the OFBiz MVC architecture. Every View in OFBiz starts with or is contained in a screen, and every screen is defined by a screen widget. An early examination of the View will eventually make the Model and Controller sections make more sense as we will learn the correct way to output data through screens that can be used throughout the later examples.

In this chapter we will be looking at:

- Equipping a webapp with a screen widget view handler.
- Creating a screen widget.
- Understanding the anatomy of a screen widget.
- Creating more complex screen widgets.
- Integrating screen widgets with FreeMarker, a powerful template engine.
- Creating screen widgets that act as templates for standard view layouts.

Equipping Our Webapp with a Screen Widget View Handler

To use screen widgets, we need to equip our webapp with a screen widget view handler. To do that, we need to set up OFBiz to listen to incoming end-user requests. So far, the requests we fired to OFBiz, like `index.html` to `webapp learning`, are processed by the embedded Tomcat server. To process screen widgets, we need to get OFBiz to intercept end-user requests and perform its special processing. Tomcat does not understand screen widgets. screen widgets are a part of the OFBiz framework.

OFBiz intercepts all end-user requests with a single entity that is called a **control servlet**. That will be explained in more detail in Chapter 8. For now, we just quickly define a control servlet to webapp learning and get on with exploring screen widgets.

In the folder \${webapp:learning}, create a folder WEB-INF. In that folder, create a new file called web.xml and into it enter the following:

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
    Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <context-param>
        <param-name>entityDelegatorName</param-name>
        <param-value>default</param-value>
    </context-param>
    <context-param>
        <param-name>localDispatcherName</param-name>
        <param-value>learning</param-value>
    </context-param>

    <filter>
        <filter-name>ContextFilter</filter-name>
        <filter-class>org.ofbiz.webapp.control.ContextFilter</filter-class>
        <init-param>
            <param-name>allowedPaths</param-name>
            <param-value>/control:/index.html</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>ContextFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>ControlServlet</servlet-name>
        <servlet-class>org.ofbiz.webapp.control.ControlServlet</servlet-
            class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ControlServlet</servlet-name>
        <url-pattern>/control/*</url-pattern>
    </servlet-mapping>
</web-app>
```

We have set the ControlServlet to intercept requests that start with control/. Any other patterns of request strings will still be handled by the embedded Tomcat server.

From here on, an OFBiz request will be taken to mean a request string with webapp name and control/ prefixed. For example, firing an OFBiz https request **SomeRequest** to webapp learning will mean a URL of `https://localhost:8443/learning/control/SomeRequest`. If the word "https" or "http" is omitted, like in "OFBiz request", it will mean a request that can be fired via either protocol.

Using the Screen Widget View Handler

Screen widgets are processed by a screen widget handler in OFBiz. In the folder `${webapp:learning}\WEB-INF`, create a new file `controller.xml`, and enter into it the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<site-conf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://www.ofbiz.org/dtds/site-conf.xsd">
    <description>Learning Module Site Configuration File</description>
    <handler name="screen" type="view"
             class="org.ofbiz.widget.screen.ScreenWidgetViewHandler"/>
</site-conf>
```

We have instructed our control servlet to handle all screen widgets with the screen widget View Handler implemented by Java class `org.ofbiz.widget.screen.ScreenWidgetViewHandler`.

Files and Locations

Screen widgets are usually defined in an OFBiz component's **widget** folder. We had already created our first screen widget in Chapter 2 in `${component:learning}\widget\partymgr\OurPartyScreens.xml`. The filename convention in OFBiz for files containing screen widgets is a string that ends with `Screens.xml`.

Creating Our First Screen Widget

Let us now create our first screen widget, the simplest screen widget possible, so we know what is the bare minimum required to make up a screen widget.

Defining a Screen Widget

Screen widgets are defined by `<screen>` elements. The name of the screen widget is determined by the `name` attribute of the `<screen>` element. The actual anatomy of a `<screen>` will be covered later in this chapter. For now, let us just quickly create our first screen widget.

In the folder \${component:learning}\widget, create a new folder named learning. In that folder, create a new file named LearningScreens.xml, and enter into it the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<screens xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation=
                  "http://www.ofbiz.org/dtds/widget-screen.xsd">
    <screen name="SimplestScreen">
        <section>
            <widgets>
                <label text="Simplest Screen possible in OFBiz!" />
            </widgets>
        </section>
    </screen>
</screens>
```

Informing the Control Servlet about the Screen Widget

Currently, the control servlet will not entertain any requests at all. For the control servlet to have any meaningful function, we need to specify a "dictionary of possible requests" in a file named controller.xml. The control servlet will look in that "dictionary" to learn about all the possible requests it should entertain in the webapp, and how. An entry in that "dictionary" is called a **request map**. We'll create our first request map in the next section. Request maps will be covered in greater detail in Chapter 8.

We now need to inform our control servlet that we have a new screen widget we wish to make accessible to end-user's requests. We first create a request map to tell our control servlet to entertain OFBiz requests with a string value of SimplestScreen.

In the file \${webapp:learning}\WEB-INF\controller.xml, just below the <handler> element, insert the following:

```
<!-- Request Mappings -->
<request-map uri="SimplestScreen">
    <response name="success" type="view" value="SimplestScreen"/>
</request-map>
<!-- end of request mappings -->
```

From here on, the name of a request map will be taken to be the value of the <request-map> element's uri attribute.

In request map `SimplestScreen`, we have specified a response called `success` that is a type `view` with a value of `SimplestScreen`. We now need to define that non-existent view. Below the `<request-map>` (after the `end of request mappings` line), insert the following:

```
<!-- View Mappings -->
<view-map name="SimplestScreen" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#SimplestScreen"/>
<!-- end of view mappings -->
```

We have pointed this new view map to the screen widget we just created. A request-map can contain any number of differently named responses depending on the outcome of events or services. By far the most common responses you will see in the request-maps, in all of the different webapp's controllers, are the responses `success` and `error`. For now, it is enough to know that the `success` response is the default.

Referencing Screen Widgets

As we can see above, the screen widget is referenced in the view-map in the `page` attribute: `component://learning/widget/learning/LearningScreens.xml#SimplestScreen`

This line contains the location to the screen widget and the location of the XML file. We can have as many `<screen>` elements in one file as we like.

Uniform Pattern of Flow in OFBiz

There is a common pattern of flow in OFBiz regarding control servlets and request handling. The first point of contact with the end - user is at the control servlet. The control servlet receives an end-user's request, and looks up its definition in the dictionary of requests to entertain (request maps defined by `<request-map>` elements in file `controller.xml`). Upon finding the request map that corresponds to the incoming request, the control servlet determines the appropriate response, which by default is the response named `success` (defined by a `<response>` element inside a `<request-map>` element). This response can point to a `view`, which leads our control servlet to look up the corresponding view map (defined by a `<view-map>` element in the file `controller.xml`). This view map in turn specifies a screen widget to be displayed as the response to the end-user.

For now, we just need to know that the flow goes from the end-user's request to a request map to a view map to a widget screen. The complete flow in OFBiz will be discussed in detail in Chapter 8.

Seeing Our First Screen Widget

Fire an OFBiz http request **SimplestScreen** to webapp learning and see our first screen widget displayed (don't forget to prefix the /control part!). This is truly the simplest screen in OFBiz, made up of plain text, not even XHTML. To see the verbatim text returned by the web server, right-click and click **View Source** (for IE) or **View Page Source** (for Firefox).

Simplest Screen possible in OFBiz!

The Anatomy of the <section> Element

The `<section>` element is the primary container for the definition of screen. There must be exactly one `<section>` element per `<screen>` element.

Our First Conditional Screen Widget

In the file `LearningScreens.xml`, insert a new screen widget:

```
<screen name="ConditionalScreen">
  <section>
    <condition>
      <if-compare field-name="parameters.show" operator="equals"
                   value="widgets"/>
    </condition>
    <actions>
      <set field="blah" value="blih"/>
    </actions>
    <widgets>
      <label text="Condition passed. Showing widgets element. Blah
                 is: ${blah}"/>
    </widgets>
    <fail-widgets>
      <label text="Condition failed! Showing fail-widgets element.
                 Blah is: ${blah}"/>
    </fail-widgets>
  </section>
</screen>
```

The contents of the `<section>` element will be explained next.

Element Order in the controller.xml

Our \${webapp:learning}\WEB-INF\controller.xml file is about to have more request maps and view maps. We must ensure that request maps and view maps are grouped together by element types and ordered. For example, all `<request-map>` elements must come before any `<view-map>` element, none after. There are 13 element types possible in a `<site-conf>` element (top-level element). Here are the ones we have met so far, listed in the order they must appear in a `controller.xml` file:

- `<include>`
- `<description>`
- `<handler>`
- `<request-map>`
- `<view-map>`

Inform the Control Servlet about the Screen Widget

We have just created our first fully decked out `<sections>` element—in our `ConditionalScreen` screen widget—showcasing the complete anatomy of the `<sections>` element. We now tell the control servlet about it.

In the folder \${webapp:learning}\WEB-INF, insert into the file `controller.xml` a new request map:

```
<request-map uri="ConditionalScreen">
<response name="success" type="view" value="ConditionalScreen"/>
</request-map>
```

and a new view map:

```
<view-map name="ConditionalScreen" type="screen"
          page="component://learning/widget/learning/
LearningScreens.xml#ConditionalScreen"/>
```

If-Then-Else Structure

The `<section>` element is made up of an if-then-else structure. This allows truly dynamic content, allowing us to conditionally decide (dynamically decide on-the-fly) to display (or hide) sections of the screen.

Let us look through this **if-then-else structure using the** `ConditionalScreen` screen widget as an example.

The <if-condition> Element

The `<condition>` element constitutes the `If` part of the `<section>` element. If the conditions in this element evaluate to true, the "then" part of the **if-then-else structure** will be processed. Otherwise, the `Else` part will be processed instead.

There are 12 possible types of conditionals we can put into the `<condition>` element. Four of them are logical operators (or joiners): `<and>`, `<xor>`, `<or>`, and `<not>`. Three are permissions-related checks: `<if-service-permission>`, `<if-has-permission>`, and `<if-entity-permission>`. The remaining five are comparators of various forms, with one of them being a simple check for empty values: `<if-validate-method>`, `<if-compare>`, `<if-compare-field>`, `<if-regexp>`, and `<if-empty>`.

We have used the simplest of those conditionals, the `<if-compare>` element. In the `<section>` element of the screen widget `ConditionalScreen`, see:

```
<condition>
<if-compare field-name="parameters.show" operator="equals"
              value="widgets"/>
</condition>
```

If the variable `parameters.show` is equal to `widgets`, the "then" part will be processed. Let's move on to the "then" part of the **if-then-else structure**.

Most of the code in the `<condition>` and `<actions>` elements is really in the OFBiz language Minilang. As you can see, it is simple to read and will be further explored. For now, we focus on exploring the anatomy of the `<section>` element.

The Then—`<actions>` and `<widgets>` Elements

If the `If` part evaluates to true, the "Then" part will be processed. The "Then" part consists of two elements: `<actions>` and `<widgets>`. The `<actions>` element is optional.

The `<actions>` Element

The `<actions>` element contains pre-processing actions, which are usually (by convention in OFBiz's MVC architecture) data manipulation, transformations, or retrieval (the "R" in CRUD). Data creation, updates and deletes (the rest of the CRUD) should be done in request map's events (explained later in the book), not in view actions (the pre-processing defined in this `<actions>` element). The View layer should only provide a view of the data, and should not change data.

There are 10 possible types of actions we can put into the `<actions>` element. Five of them are data retrieval (from the database) actions which place data into variables, variables that are accessible and we can use in the screen widget's context. These five action types are: `<entity-and>`, `<entity-condition>`, `<entity-one>`, `<get-related-one>`, and `<get-related>`. We will understand these more after Chapter 7.

Two of the 10 possible action types invoke code that could be complex business logic. The action type `<script>` invokes codes written using BeanShell. The action type `<service>` invokes codes wrapped in an OFBiz service (a major benefit of OFBiz and explained later in the book).

Three of the 10 possible action types simply place values into variables, in the same way as the data retrieval actions. The action types `<property-map>` and `<property-to-field>` retrieve values from a property or configuration file. The simplest action of all, `<set>`, places any value we want and assigns it any variable. We will use this in our example now.

We have used the simplest action in our screen widget `ConditionalScreen`:

```
<actions>
<set field="blah" value="blih"/>
</actions>
```

We have assigned the string value `blih` to variable `blah`.

Note that the `<actions>` element is not processed until after the `<condition>` is completely processed. Therefore, the value `blah` is still not set if assessed in the `<condition>`. The following will evaluate to false:

```
<condition>
<if-compare field="blah" operator="equals" value="blih"/>
</condition>
```

The `<widgets>` Element

The `<widgets>` element is a display element, in which we place content we want displayed. There are 15 types of elements we can put inside a `<widgets>` element. Most of those element types are complex, involving nested structures of other widget artifacts, some are even recursive (think "screen within screen within screen"). Four of those element types are simple: `<container>`, `<image>`, `<link>`, and `<label>`.

All 15 elements will be described in detail later in the book. Briefly for now, for those of us who know XHTML, we just need to know that the `<container>` element corresponds to the XHTML element `<div>`, the `<image>` element to the `` element, the `<link>` to the `<a>`, and the `<label>` to plain text (non-markup) in XHTML.

We have used the simple `<label>` element in our `ConditionalScreen`:

```
<widgets>
<label text="Condition passed. Showing widgets element. Blah is:
    ${blah}"/>
</widgets>
```

If the `If` part evaluates to true, we should be able to see the text: **Condition passed. Showing widgets element. Blah is: blih.**

The Else—`<fail-widgets>` Element

The `<fail-widgets>` element is exactly like the `<widgets>` element in structure and function. However, the `<fail-widgets>` element is only processed if the `If` part evaluates to false. In that case, the "Then" part is completely skipped, and neither the `<actions>` element nor the `<widgets>` element are processed at all.

Again, we have used the simple `<label>` element in our `ConditionalScreen`:

```
<fail-widgets>
<label text="Condition failed! Showing fail-widgets element. Blah is:
    ${blah}"/>
</fail-widgets>
```

If the `If` part evaluates to false, we should be able to see the text **Condition failed! Showing fail-widgets element. Blah is:** Note how the `<actions>` will be skipped, and the variable `blah` will be empty (not set).

The Minimum `<section>`

Note from our first screen widget `SimplestScreen` that the bare minimum `<section>` element must contain at least the `<widget>` sub element. That is, if the `If` part is omitted, it can only be presumed "true". In that case, we process the "Then" part. The `Else` part won't be necessary in this case.

Sending Parameters with Requests

To play with our `ConditionalScreen` screen widget, we will need to send **request parameters** with the requests we fire to webapp `learning`. An obvious and easy way to do so is to send GET parameters. GET parameters are sent in the URL of the request, after a "?" character. Request parameters are specified in key-value pairs, with the operator "=" between the key and the value, like this:

```
someKey=someValue
```

If there are more than one key-value pairs, they are delimited by an "&" character, like this:

```
keyOne=valueOne&keyTwo=valueTwo
```

From here on, when we fire say an OFBiz http request `SomeRequest` with parameters `paramOne=valueOne` and `paramTwo=valueTwo` to webapp `learning`, we will mean a URL of `http://localhost:8080/learning/control/SomeRequest?paramOne=valueOne¶mTwo=valueTwo`. It should be easier to read the former, rather than the long URL.

POST parameters are sent via forms, not URLs, and will be covered in the next chapter on *Form Widgets*.

Seeing Our Conditional Screen Widget

We can see our new screen widget by firing an OFBiz http request `ConditionalScreen` with parameter `show=widgets` to webapp `learning`, that is, `http://localhost:8080/learning/control/ConditionalScreen?show=widgets`.

We will see the "Then" part being displayed.

To see the `Else` part instead, set the parameter `show` to a value of anything other than `widgets`, say `fail`. Notice how the variable `blah` is not set to `blah`, but is empty instead. This shows that the `<actions>` element is truly an integral part of the `Else` portion of the `if-then-else`. The elements `<actions>` and `<widgets>` are either both processed or not at all.

Screen Widget Context and Variables

Each screen widget's actions (in `<actions>` sub-element) operate within their own private context. In that context, all variables are private to that context. We'll cover more on multiple and nested contexts later, since we're still dealing with single standalone screen widgets.

The examples we look at here will be using the Java language in the BeanShell framework. Minilang is a further simplification of OFBiz-specific functions written in Java. Hence, all Minilang functions have Java counterparts. It is good to learn first the fundamental form (Java) of OFBiz functions; Minilang is often a mirror image of that form and can be easily picked up afterwards. Moreover, screen widget actions often fall back to BeanShell wherever Minilang is inadequate.

Necessary Java concepts will be explained along the way. Java is a well-structured language that can be quite easily and intuitively understood. Minilang counterparts will be highlighted too, if they exist.

Utility Objects in Context

The context contains a few objects, placed there for our convenience. These objects are important utility objects:

- `screens` is an `org.ofbiz.widget.screen.ScreenRenderer` object
- `globalContext` is an `org.ofbiz.base.util.collections.MapStack` object
- `nullField` is an `org.ofbiz.entity.GenericEntity$NullField` object
- `availableLocales` is a `java.util.List` object
- `locale` is a `java.util.Locale` object
- `delegator` is an `org.ofbiz.entity.GenericDelegator` object
- `dispatcher` is an `org.ofbiz.service.GenericDispatcher` object
- `security` is an `org.ofbiz.security.OFBizSecurity` object
- `userLogin` is an `org.ofbiz.entity.GenericValue` object
- `parameters` is an `java.util.Map` object

Utility Objects—Brief Explanation

Object `screens` are used in FreeMarker templates to render sub-screens, defined by screen widgets.

Object `globalContext` is the one single shared context that all nested screens within a screen (including the top-level screen itself) can see and use.

Object `nullField` is used as a null value for database checks that involve checking for null values in database fields. Null is computer speak for empty, no value. We'll cover this in the Minilang chapter. This isn't used for BeanShell.

Object `availableLocales` is a list of available locales or translations we want OFBiz to support. The list is defined in the file `${component : common} \config\general.properties` property `locales.available` (line 30). That property is commented out now (by the "#" character), so the list will be every installed locale setting for our Windows OS. This is almost never desirable, since OFBiz may not have a translation for every locale setting our Windows OS has installed. Fire an http request to webapp ecommerce to see the possibly huge list of available languages in the top-right drop-down list, assuming your installation of Windows has that many locale settings installed. Only a sub set of those locales have translations in OFBiz; wherever missing translations are encountered, English is used.

Object `locale` is the current selected locale. Based on this value, OFBiz determines which language to display screens in. This value is pulled from the end-user's current session with OFBiz, in a session attribute named `locale`. If it isn't found there, the end-user's last login's chosen locale will be used. If that value is not found, the end-user's browser's language setting is used. As a last resort, the locale specified in the file `${OFBizInstallFolder} \startofbiz.bat` string `-Duser.language=en` is looked up, which in this case means English (en). This last feature was introduced to deal with a problem in the Sun Java API and how it deals with properties files. It has been removed in recent OFBiz versions since OFBiz is now using XML instead of properties files (ResourceBundle) to store its translations.

Object `delegator` is used for communicating with the database.

Object `dispatcher` is used for calling on OFBiz services.

Object `security` provides a number of convenient tools for security-related and permissions checks.

Object `userLogin`, if it exists, contains the details of the logged in end - user. It is tagged on to the session as attribute `userLogin`. This is often used in conjunction with the security object.

Object `parameters` is a map of request parameters. A map, in computing speak, is a collection of key-value pairs. Actually, parameters consist of a combination of request attributes, request parameters, session attributes, and ContextServlet attributes.

We will be exploring the `parameters` and `globalContext` objects now; the others will be covered later.

The Local (Private) Context

Accessing the local context of a screen widget is straightforward and automatic in the `<actions>` sub-element (or in Minilang, rather). Simply assigning a value to a variable will mean creating the variable (or modifying it if it already exists) in the local context. An example we saw is this:

```
<set field="blah" value="blih"/>
```

Let us now look at the BeanShell equivalent. In `LearningScreens.xml`, insert a new screen widget `ConditionalScreenWithBsh`:

```
<screen name="ConditionalScreenWithBsh">
  <section>
    <condition>
      <if-compare field-name="parameters.show" operator="equals"
                   value="widgets"/>
    </condition>
    <actions>
      <script location="component://learning/webapp/learning/WEB-
                     INF/actions/learning/conditionalScreenActions.bsh"/>
    </actions>
    <widgets>
      <label text="Condition passed. Showing widgets element. Blah
                  is: ${blah}"/>
    </widgets>
    <fail-widgets>
      <label text="Condition failed! Showing fail-widgets element.
                  Blah is: ${blah}"/>
    </fail-widgets>
  </section>
</screen>
```

In the folder `${webapp:learning}\WEB-INF` insert into `controller.xml` a new request map:

```
<request-map uri="ConditionalScreenWithBsh">
  <response name="success" type="view"
            value="ConditionalScreenWithBsh"/>
</request-map>
```

and a new view map:

```
<view-map name="ConditionalScreenWithBsh" type="screen"
          page="component://learning/widget/learning/
                LearningScreens.xml#ConditionalScreenWithBsh"/>
```

In the folder \${webapp:learning}\WEB-INF create new folders actions\learning.
Insert into a new file called conditionalScreenActions.bsh the following:

```
String blah = "blih";
```

Loosely Typed Variables in BeanShell

In BeanShell, variables can be "loosely typed". Loosely typed variables have no defined type until an assignment of value is made. The variable then adopts the type of the value assigned. Variable `blah` has type `java.lang.String` because the value "blih" (in double quotes) is a literal of type `String`. Although you may see this in many places in existing BeanShell code, it is not a good practice to keep. Note though that in recent OFBiz versions, where Groovy is replacing Beanshell, this practice is common.

Literals Versus Variables

Literals are literal values, as opposed to variables that are "containers" that can hold variable content or values. A numerical literal would be written like `123` (without the double quotes). Putting `blah = 123;` would mean changing the type of variable `blah` into an `int` (Java class for integers).

At first glance, our `conditionalScreenActions.bsh` seems to be the equivalent of `<set field="blah" value="blih"/>`. But it is not. Firing an OFBiz http request `ConditionalScreenWithBsh` with parameter `show=widgets` to `webapp learning` will not yield the correct result. The variable `blah` variable `blah` is still empty even though `<widgets>` element is processed. Let us see why.

When OFBiz's screen widget handler encounters a `<script>`, which must invariably point to a `.bsh` file (no other file types supported), it creates an environment in which to run the BeanShell script.

This BeanShell environment can have variables like in the `<actions>` sub-element, but there is a big difference. In the BeanShell environment, for any variables that are newly created or any variables that are assigned new values, those variables' changes will not be carried over to the screen widget context.

The screen widget handler places a special object named `context` of type `MapStack` into the BeanShell environment. The object is used for manipulating and creating variables in a screen widget's local context. Add the following line to the bottom of the `conditionalScreenActions.bsh` file:

```
context.put("blah", blah);
```

From here on, the special BeanShell environment created by the OFBiz Widget Engine for screen widgets and any other widgets (covered in later chapters) will be termed **widget BeanShell environment**. The special way to do BeanShell in that environment will be termed **widget BeanShell**. For example, in widget BeanShell, we use the `context` object.

A refresh of the browser window should now yield the expected results.

Accessing Variables' Values

In Minilang, getting the value of a variable involves the literal form " `${<variable-name>}` ". An example is `<label text="Condition passed. Showing widgets element. Blah is: ${blah}" />`.

In widget BeanShell, it is `context.get("blah")`.

Accessing Map Variables' Values

In Minilang, accessing a Map variable's values involves the literal form " `${<variable-name>. <key-name>}` ". For example, `parameters.show` gets, from object "parameters", the value for the key-value pair that has a key of "show".

In widget BeanShell, the equivalent of the above example is `parameters.get("show")`.

To test this, rewrite screen widget `ConditionalScreenWithBsh` into:

```
<screen name="ConditionalScreenWithBsh">
    <section>
        <condition>
            <if-compare field-name="parameters.show" operator="equals"
                         value="widgets"/>
        </condition>
        <actions>
            <script location="component://learning/webapp/learning/WEB-INF/
                           actions/learning/conditionalScreenActions.bsh"/>
        </actions>
        <widgets>
            <label text=" ${passMsg} "/>
        </widgets>
        <fail-widgets>
            <label text="Condition failed! Showing fail-widgets element.
                         Blah is: ${blah}"/>
        </fail-widgets>
    </section>
</screen>
```

and rewrite `ConditionalScreenActions.bsh` into:

```
context.put("blah", "blih");
passMsg = "Condition passed. Showing widgets element.<br/>" +
    "Blah is: " + context.get("blah") + "<br/>" +
    "show is: " + parameters.get("show");
context.put("passMsg", passMsg);
```

In Java, the two key methods of Map objects we will be using are very often `put` and `get`. The `put` method inserts a new key-value pair into a map, or updates the value of an existing pair if the key already exists. The `get` method retrieves the value of a key-value pair in the map. This applies not just to widget BeanShell, but to Java in general as well.

The Global Context

Like the local context, accessing the global context of a screen widget is straightforward and automatic in the `<actions>` subelement.

In widget BeanShell, we use the object `globalContext` in the same way we would use the object `context`:

```
globalContext.put("someGlobalVariabel", "We can see this in every
nested screen, as well as in every parent screen.");
```

We will play with this more when we cover nested screens later in this chapter.

Nested Sections for Nested Conditions

The `<section>` element can be nested in a `<widgets>` or a `<fail-widgets>` element, in the "then" and "else" of the **if-then-else, respectively**. Specifically, within those "then" and "else" parts, it can also be contained in a `<container>` element, a `<decorator-section>` element or a `<html-template-decorator-section>` (the last two elements to be discussed later in the book).

In the short, the `<section>` element can be nested inside either the "then" part or the "else" part to create a nested conditional, a whole **if-then-else structure nested inside the outer if-then-else**.

In the file `LearningScreens.xml`, create a screen with nested sections like so:

```
<screen name="NestedSections">
<section>
    <condition>
        <if-compare field-name="parameters.outer" operator="equals"
            value="true"/>
```

Screen Widgets

```
</condition>
<actions>
    <set field="blah" value="blih"/>
</actions>
<widgets>
    <container><label text="In Then part. Blah is:
        ${blah}" /></container>
    <container><label text="inner is:
        ${parameters.inner}" /></container>
<section>
    <condition>
        <if-compare field-name="parameters.inner" operator="equals"
            value="true"/>
    </condition>
    <actions>
        <set field="inner.blah" value="thenBlih"/>
    </actions>
    <widgets>
        <label text="In Then-Then part. inner.blah is:
            ${inner.blah}" />
    </widgets>
    <fail-widgets>
        <label text="In Then-Else part. inner.blah is:
            ${inner.blah}" />
    </fail-widgets>
    </section>
</widgets>
<fail-widgets>
    <container><label text="In Else part. Blah is:
        ${blah}" /></container>
    <container><label text="inner is:
        ${parameters.inner}" /></container>
<section>
    <condition>
        <if-compare field-name="parameters.inner" operator="equals"
            value="true"/>
    </condition>
    <actions>
        <set field="inner.blah" value="elseBlih"/>
    </actions>
    <widgets>
        <label text="In Else-Then part. inner.blah is:
            ${inner.blah}" />
    </widgets>
    <fail-widgets>
```

```
<label text="In Else-Else part. inner.blah is:  
    ${inner.blah}"/>  
</fail-widgets>  
</section>  
</fail-widgets>  
</section>  
</screen>
```

In the file controller.xml, create the request map and view map like this:

```
<request-map uri="NestedSections">  
<response name="success" type="view" value="NestedSections"/>  
</request-map>
```

and this:

```
<view-map name="NestedSections" type="screen"  
page="component://learning/widget/learning/  
LearningScreens.xml#NestedSections"/>
```

Fire an OFBiz http request **NestedSections** with parameters outer and inner to webapp learning. Those parameter's values can be true or false. Try various combinations, such as true and true or false and false.

Variables outer and inner are both true:

In Then part. Blah is: blih
inner is: true
In Then-Then part. inner.blah is: thenBlih

Variable outer is true and inner is false:

In Then part. Blah is: blih
inner is: false
In Then-Else part. inner.blah is:

Variable outer is false and inner is true:

In Else part. Blah is:
inner is: true
In Else-Then part. inner.blah is: elseBlih

Variables `outer` and `inner` are both `false`:

In Else part. Blah is:
`inner` is: `false`
In Else-Else part. `inner.blah` is:

Organizing a Large Screen into Smaller Screens

One of the most useful features in screen widgets is the ability to organize a large screen into smaller screens (often termed "screenlets"). To organize a large screen into smaller ones, we simply break up the display element of the large screen into smaller screens. As mentioned previously in this chapter, the display elements of a screen are the `<widgets>` and `<fail-widgets>` elements. In those elements, we can include other screen widgets.

The concept used here is **includers**, widget elements that include smaller screens into a larger screen. Of the 15 types of elements we can put inside a `<widgets>` or `<fail-widgets>` element, nine are classified as includers. We will look at one of those, the `<include-screen>` element.

We had already created a simple screen widget named `SimplestScreen`. Let us now create another simple screen widget named `Another SimpleScreen`. Edit `LearningScreens.xml`, and insert the following screen widget:

```
<screen name="AnotherSimpleScreen">
<section>
  <widgets>
    <label text="Just Another Simple Screen."/>
  </widgets>
</section>
</screen>
```

There is no need to tie that screen widget to our control servlet via a request map and a view map in `controller.xml`. We won't be allowing end-users to access that screen widget directly.

We next create a compounded screen widget that includes the two smaller screens into one whole. Enter into `LearningScreens.xml` a new screen widget named `CompoundedScreen`:

```
<screen name="CompoundedScreen">
<section>
  <widgets>
    <container><include-screen name="SimplestScreen"/></container>
    <container><include-screen name="AnotherSimpleScreen"/>
    </container>
  </widgets>
</section>
</screen>
```

In the folder `${webapp:learning}\WEB-INF`, insert into `controller.xml` a new request map:

```
<request-map uri="CompoundedScreen">
<response name="success" type="view" value="CompoundedScreen"/>
</request-map>
```

and a new view map:

```
<view-map name="CompoundedScreen" type="screen"
  page="component://learning/widget/learning/
  LearningScreens.xml#CompoundedScreen"/>
```

Fire an OFBiz http request `CompoundedScreen` to `webapp learning` to see this compounded screen. Compounded screens are placed one after another, so screen widget `AnotherSimpleScreen` should be right below `SimplestScreen`. Right-click and click **View Page Source** to see the verbatim text to confirm this.

In the above example, the included screens are in the same file (`LearningScreens.xml`) as the including screen `CompoundedScreen`. If the included screens are in another file, the `<include-screen>` element can have an optional `location` attribute that contains a component URL that points to the other file. The same principle applies for many other screen widget sub elements; they can also have the optional `location` attribute. For an example, take a look at some of the order manager screen widgets (located in `applications\order\widgets`), in particular `OrderEntryCommonScreens.xml` line 76:

```
<include-screen name="orderHeaderInfo"
  location="component://order/widget/ordermgr/
  OrderEntryCartScreens.xml"/>
```

The Global Context Revisited

We will now explore the global context for screen widgets with concrete examples. In `LearningScreens.xml`, insert a new screen widget `GlobalContext`:

```
<screen name="GlobalContext">
  <section>
    <actions>
      <set field="global1" value="Global Value A" global="true"/>
      <set field="local1" value="Local Value a"/>
      <script location="component://learning/webapp/learning/
          WEB-INF/actions/learning/globalContext.bsh"/>
    </actions>
    <widgets>
      <label text="&lt;table
          style="width:50%"&gt;&lt;tr&gt;">
        <label text="&lt;td style="border:thin solid"&gt;">
          <include-screen name="NestedScreen"/>
        <label text="&lt;/td&gt;">
        <label text="&lt;td style="border:thin solid"&gt;">
          <container><label text="GlobalContext screen:"/></container>
          <container><label text="global1: ${global1}" /></container>
          <container><label text="global2: ${global2}" /></container>
          <container><label text="global3: ${global3}" /></container>
          <container><label text="global4: ${global4}" /></container>
          <container><label text="local1: ${local1}" /></container>
          <container><label text="local2: ${local2}" /></container>
          <container><label text="local3: ${local3}" /></container>
          <container><label text="local4: ${local4}" /></container>
        <label text="&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;">
      </widgets>
    </section>
  </screen>
```

and a new Screen widget `NestedScreen`:

```
<screen name="NestedScreen">
  <section>
    <actions>
      <set field="global3" value="Global Value C" global="true"/>
      <set field="global4" value="Global Value D" global="true"/>
      <set field="local3" value="Local Value c"/>
      <set field="local4" value="Local Value d"/>
    </actions>
    <widgets>
```

```
<container><label text="NestedScreen screen:"/></container>
<container><label text="global1: ${global1}" /></container>
<container><label text="global2: ${global2}" /></container>
<container><label text="global3: ${global3}" /></container>
<container><label text="global4: ${global4}" /></container>
<container><label text="local1: ${local1}" /></container>
<container><label text="local2: ${local2}" /></container>
<container><label text="local3: ${local3}" /></container>
<container><label text="local4: ${local4}" /></container>
</widgets>
</section>
</screen>
```

Screen GlobalContext has a few XHTML elements hacked in via `<label>` elements, specifically the XHTML `<table>` and related elements. OFBiz widgets aren't a complete replacement for XHTML; they aren't complete reinventions of a perfectly good wheel that is XHTML. Later in this chapter, we will explore the proper ways to integrate XHTML (via FreeMarker) with OFBiz widgets. For now, the `<label>` hack is needed.

In the file `${webapp:learning}\WEB-INF\controller.xml`, insert a new request map:

```
<request-map uri="GlobalContext">
    <response name="success" type="view" value="GlobalContext"/>
</request-map>
```

and a new view map:

```
<view-map name="GlobalContext" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#GlobalContext"/>
```

In the folder `${webapp:learning}\WEB-INF\learning\actions\learning`, create a new file `globalContext.bsh` and enter into it this:

```
globalContext = context.get("globalContext");
globalContext.put("global2", "Global Value B");
context.put("local2", "Local Value b");
```

Fire an OFBiz http request `globalContext` to webapp learning.

NestedScreen screen: global1: Global Value A global2: Global Value B global3: Global Value C global4: Global Value D local1: Local Value a local2: Local Value b local3: Local Value c local4: Local Value d	GlobalContext screen: global1: Global Value A global2: Global Value B global3: Global Value C global4: Global Value D local1: Local Value a local2: Local Value b local3: local4:
--	---

Note how the global variables are visible to both the outer and the inner (nested) screens. The inner screen's variables are not visible to the outer screen. However, the converse is not true, which brings us to an interesting feature in screen widget contexts for nested screens.

Outer Contexts Visible to Nested Ones

Outer contexts are visible to inner contexts, unless overridden by variables with similar names in the inner contexts. In the above example with screens `GlobalContext` and `NestedScreen`, that much is obvious.

Now, let's try overriding variables in the outer context. In the screen `NestedScreen`, replace the `<actions>` sub element with this:

```
<actions>
  <set field="global3" value="Global Value C" global="true"/>
  <set field="global4" value="Global Value D" global="true"/>
  <set field="local3" value="Local Value c"/>
  <set field="local4" value="Local Value d"/>
  <set field="local1" value="Local Value z"/>
</actions>
```

Fire an OFBiz http request `globalContext` to webapp learning. Note how the variable access `${local1}`, when being performed from screen `NestedScreen`, no longer looks up variable `local1` in the outer context, but in the nested context instead.

To summarize, a variable access will attempt to look up the variable in the local context, failing which it will attempt to find the most specific context that contains that variable, going upwards into the outer contexts. A three-level nested screen will better showcase this feature. That is left as an exercise for you, if you are looking to confirm this feature.

The same feature and principle of operation applies for decorator screens which are covered towards the end of this chapter.

Screen Widget's Integration with FreeMarker

Very often, we will use the screen widget's integration with FreeMarker. At this point, it is not yet necessary to understand FreeMarker itself.

FreeMarker is a technology that allows conditional scripting in XHTML, much like how the screen widget also has conditionals in an **if-then-else structure**. FreeMarker's conditional scripting is a lot more powerful than that. Wherever screen widgets are lacking, in their **if-then-else structure for conditional scripting or their presentation elements**, we fall back on FreeMarker.

FreeMarker documents have (and should have) filenames that end with `.ftl`.

The screen widget construct that includes a FreeMarker document is like this:

```
<platform-specific><html>
<html-template location="<path-to-some-ftl-file>" />
</html></platform-specific>
```

where "`<path-to-some-ftl-file>`" specifies where to find the `.ftl` to include.

An example is coming up in the next section.

The `location` attribute of element `<html-template>` must point to an `.ftl` file. The reason why screen widgets only support integration with FreeMarker documents and not XHTML documents (with filenames that end with `.html`) is because FreeMarker documents are really XHTML documents if we omit all FreeMarker script constructs. In that sense, a screen widget does in practice have the ability to include (integrate with) raw XHTML contents.

screen widgets have been designed for speed of use and ease of development. They have not been designed to handle large amounts of logic or a complicated output. No matter how complicated the overall design of the page is, the screen widgets can be used to break the screen down into different components – how much you use them to break up the screen is up to you, and experience will tell you to what extent to use screen widgets. There is no better example to show how screen widgets work and how FreeMarker is used with the Screens than the ecommerce component.

We have seen enough of OFBiz now to start trying to piece together how to get from A to B in the code. Open up the ecommerce component in your browser by firing an http request to the ecommerce webapp. Have a click around and see if you can follow your click through the controller.xml and into the screen widget. From there, see if you can piece together any FreeMarker templates and BeanShell actions that are used.

Should you stumble upon the \${mainDecoratorLocation} variable in any of the Screens.xml files, this location is specified in the webapp's web.xml file.

Cleaning Up in the "party" Component

We take the opportunity here to clean up the mess we made in the party component in Chapter 2. Edit \${component:learning}\widget\partymgr\OurPartyScreens.xml and insert a new screen widget:

```
<screen name="editcontactmech.extend">
<section>
<widgets>
<platform-specific><html>
<html-templatelocation="component://learning/webapp/
partymgr/party/editcontactmech.extend.ftl"/>
</html></platform-specific>
</widgets>
</section>
</screen>
```

In the folder \${component:learning}\webapp\partymgr\party, create a new file editcontactmech.extend.ftl. Enter into the file this:

```
<tr>
<td class="label">Planetoo</td>
<td>
<input type="text" size="30" maxlength="100"
name="planet" value="${(mechMap.postalAddress.planet)!!}">
</td>
</tr>
```

Edit the file \${webapp:partymgr}\party\editcontactmech.ftl, go to line 166 and replace:

```
<tr>
<td class="label">Planet</td>
<td>
<input type="text" size="30" maxlength="100"
```

```

        name="planet" value="${(mechMap.postalAddress.planet)!!}">
</td>
</tr>
```

with:

```

${screens.render("component://learning/widget/partymgr/
OurPartyScreens.xml#editcontactmech.extend")}
```

Recall that the `screens` object in the context of a screen widget is used to render sub-screens.

Accessing variables in FreeMarker is the same as doing so in Minilang, using the form `${<variable-name>}`. In the above example, we are calling the instance method `render` of Java class `org.ofbiz.widget.screen.ScreenRenderer`, which takes in a single parameter of type `String` that is the physical location of the screen widget to include.

We have just included a sub-screen `editcontactmech.extend` into the file `editcontactmech.ftl`. The method of inclusion here is actually the same as using `<include-screen>` inside of a screen widget, although the syntax is different. The former is in FreeMarker syntax while the latter is in screen widget syntax.

To see the above in action, fire an http request to webapp `partymgr`, log in, find a party with the last name of **Researcher**, and first name **OFBiz** (or any other party with a postal address that can be updated). Bring up the update screen for the party's postal address to see the field **Planet** with a misspelled label of **Planetoo**.

State/Province	California
Zip/Postal Code	90210
Country	United States
Planetoo	Earth
Allow Solicitation?	<input checked="" type="checkbox"/>
Go Back Save	

Correct the misspelling in file `editcontactmech.extend.ftl` and refresh the browser to see it take effect.

The file `editcontactmech.ftl` now has a smaller footprint of change. The bulk of the change (or extension code) is in file `editcontactmech.extend.ftl`.

Commenting Changes to the Core Code

We are left with a small, but unavoidable situation, whereby some of our project specific code is left in the core OFBiz code. By the time our project is complete, we may end up with hundreds or even thousands of these small additions. In time, OFBiz 5.0 will be released and we may wish to upgrade. For the upgrade to be successful, as well as copying our whole component over into the new hot-deploy, we will have to go through all of the changes to the core code we have modified and "port" these changes over to the new version. This painful task would be virtually impossible if we do not have a way of marking these changes. A way that can be searched quickly, so we can produce a complete list of changes in a few seconds, rather than trawling through every line of code, trying to remember the changes, or comparing each file with the repository. This can be done quite simply by commenting out the old code and placing a comment above the new code. If the old code remains where it is, changing back will be quick and easy.

This can be done in any type of file, be it Java, BeanShell, FTL or XML by using the language's comment syntax. So

Old Code

would Become in Java/BeanShell:

```
// Begin Project Name Specific: Developer - Date - Description
New Code
/* Old Code */
//End Project Name Specific: Developer - Date - Description
```

XML:

```
<!-- Begin Project Name Specific: Developer - Date - Description -->
New Code
<!-- Old Code -->
<!--End Project Name Specific: Developer - Date - Description -->
```

and FreeMarker:

```
<!-- Begin Project Name Specific: Developer - Date - Description-->
New Code
<!-- Old Code -->
<!--End Project Name Specific: Developer - Date - Description -->
```

So our most recent addition to the `editcontactmech.extend.ftl` file should now look like this:

```
<!-- Begin Learning Specific: My Name - 01/01/2009 - Added planet text
field to form -->
```

```
 ${screens.render("component://learning/widget/partymgr/
OurPartyScreens.xml#editcontactmech.extend")}

<!-- -->
<!-- End Learning Specific: My Name - 01/01/2009 - Added planet text
field to form -->
```

Note the addition of the empty comments to make it absolutely clear that this is an addition and no code was actually changed.

If you are managing a team of developers, this should be strictly enforced if you intend to upgrade at any time in the future. By adding the developer name, date, and description or reason for the change, other developers working within the team can quickly see what implications this change has. It will also be useful information for whoever is performing the upgrade.

Screen Widgets as Templates

The fact that we can include smaller screens inside larger screens will lead us to this next logical step—the ability to create templates with "slots" that can include any arbitrary sub-screens.

A Candidate for Templating

We will now look for an issue for which we can and should apply templating.

Since OFBiz is a web-based application, all output and user-interfaces should be in XHTML format. The previous screen widget examples we had toyed with in this chapter do not output well-formed XHTML documents. Let us now fix that problem.

A well-formed XHTML document consists of a document type definition (DTD) declaration and a top-level element `<html>`. The `<html>` element must contain two elements `<head>` and `<body>`. The `<head>` element must contain a `<title>` element. That means every XHTML content must at least be surrounded by a header:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>OFBiz XHTML Document</title>
</head>
<body>
```

and a footer:

```
</body>
</html>
```

Creating the Header

In the folder \${webapp:learning}, create a new folder: includes. In that folder, create a file header.ftl and enter into it the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>OFBiz XHTML Document</title>
</head>
<body>
```

Insert a screen widget called header into LearningScreens.xml like this:

```
<screen name="header">
<section>
<widgets>
<platform-specific><html>
<html-template location="component://learning/webapp/learning/
includes/header.ftl"/>
</html></platform-specific>
</widgets>
</section>
</screen>
```

Creating the Footer

In the includes folder in \${webapp:learning}, create a file footer.ftl and enter into it the following:

```
</body>
</html>
```

Insert a screen widget called footer into LearningScreens.xml like this:

```
<screen name="footer">
<section>
<widgets>
<platform-specific><html>
<html-template location="component://learning/webapp/learning/
includes/footer.ftl"/>
</html></platform-specific>
</widgets>
</section>
</screen>
```

Using Our Header and Footer

In the file `LearningScreens.xml`, edit the screen widget `CompoundedScreen` and surround the two `<include-screen>` elements with the header and footer like this:

```
<include-screen name="header"/>
<include-screen name="SimplestScreen"/>
<include-screen name="AnotherSimpleScreen"/>
<include-screen name="footer"/>
```

Seeing Our First Well-Formed XHTML Document

Fire an OFBiz http request `CompoundedScreen` to `webapp learning`, view the page source, and see our first well-formed XHTML document!

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD X
2 <html>
3 <head>
4 <title>OFBiz XHTML Document</title>
5 </head>
6 <body>
7 Simplest Screen possible in OFBiz!
8 Just Another Simple Screen.
9 </body>
10 </html>
```

The displayed screen looks the same as before because most browsers can correct (to some extent) malformed XHTML documents.

It is now obvious that we will need to include the XHTML header and footer in every XHTML document OFBiz displays. That means every view we create must have the same header and footer. Here, we must make the distinction between a view, which relates to a complete screen widget, and an individual screen widget, which may itself be just a sub-screen of a larger complete screen. Only views need those XHTML header and footer, not the sub-screens.

We won't have to manually include the header and footer into every view. Such a manual task could introduce human errors, such as getting the header-content-footer order wrong. We need a templating system.

Using Decorator Screen Widgets for Templating

OFBiz uses decorator screen widgets for view templating. The word "decorator" here comes from the design pattern it is based on—the Decorator Pattern, where the decorator wraps itself around the object to be decorated. This design pattern lends itself easily to our templating needs, because a decorator will act as a template with "slots" in which we can insert (include, in screen widget terminology) content. The decorator wraps itself around chunks of content, so to speak.

Creating a XHTML Decorator Screen

Insert into the file `LearningScreens.xml` a new screen widget named `xhtml-decorator`:

```
<screen name="xhtml-decorator">
<section>
  <widgets>
    <include-screen name="header"/>
    <decorator-section-include name="body"/>
    <include-screen name="footer"/>
  </widgets>
</section>
</screen>
```

Note how similar this is to the screen widget `CompoundedScreen`, except that the entire content is replaced by the `<decorator-section-include>` element. The `<decorator-section-include>` is like a named content slot. That element tells the decorator to insert the content section named `body` right between the header and the footer.

The fact that content sections can have names means that we can have multiple "slots" in a decorator that contain different content sections. In short, a decorator screen can specify named content slots in which similarly named content sections can be inserted or included. For now, let's stick to a single-slot decorator in the following examples.

Using the XHTML Decorator Screen

In the file `LearningScreens.xml`, edit the screen widget `CompoundedScreen` to use the screen widget `xhtml-decorator`:

```
<screen name="CompoundedScreen">
<section>
  <widgets>
```

```

<decorator-screen name="xhtml-decorator">
  <decorator-section name="body">
    <include-screen name="SimplestScreen"/>
    <include-screen name="AnotherSimpleScreen"/>
  </decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>

```

The `<decorator-screen>` element indicates that we want to apply the decorator screen `xhtml-decorator` to the content included in the `<decorator-screen>` element.

In order for a `<decorator-screen>` element to have any effect at all, it needs to specify named content sections with the `<decorator-section>` element. There can be multiple named content sections, although the above example only shows one. The named content sections are substituted into the decorator's named content slots.

The named content section `body` in the above example is a combination of two simple screens. These two simple screens will be inserted into the decorator's named content slot `body`.

Seeing Our First Decorator in Action

Fire an OFBiz http request `CompoundedScreen` to `webapp learning`. View the page source to confirm that it is the same well-formed XHTML document we saw in the section called *A Candidate for Templating*.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD X
2  <html>
3  <head>
4  <title>OFBiz XHTML Document</title>
5  </head>
6  <body>
7  Simplest Screen possible in OFBiz!
8  Just Another Simple Screen.
9  </body>
10 </html>

```

The page source shows the same content as our first well-formed version of `CompoundedScreen`. We had used a single decorator (`xhtml-decorator`) to replace two `<include-screen>` elements, elements which served to build the header and footer of our well-formed XHTML document.

Multiple Content Slots

Our decorator named `xhtml-decorator` still isn't perfect, the `<title>` element in `header.ftl` is supposed to be variable, but it currently has a fixed value of **OFBiz XHTML Document**. That means our decorator `xhtml-decorator` needs two named content slots: one named `body` and another named `title`.

We need to split up our existing `header.ftl` to create a new content slot.

Creating the First Half of the Header

In the folder `${webapp:learning}\includes`, create a file `header1.ftl` and enter into it the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>
```

In the file `LearningScreens.xml`, create a new screen widget `header1` like this:

```
<screen name="header1">
<section>
  <widgets>
    <platform-specific><html>
      <html-template location="component://learning/webapp/learning/
        includes/header1.ftl"/>
    </html></platform-specific>
  </widgets>
</section>
</screen>
```

Creating the Second Half of the Header

In the folder `${webapp:learning}\includes`, create a file `header2.ftl` and enter into it the following:

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

In the file `LearningScreens.xml`, create a new screen widget `header2` like this:

```
<screen name="header2">
<section>
  <widgets>
    <platform-specific><html>
      <html-template location="component://learning/webapp/learning/
        includes/header2.ftl"/>
    </html></platform-specific>
  </widgets>
</section>
</screen>
```

Adding a Content Slot to a Decorator

In the file `LearningScreens.xml`, look at the decorator `xhtml-decorator`, edit the `<widgets>` element's contents like this:

```
<include-screen name="header1"/>
<decorator-section-include name="title"/>
<include-screen name="header2"/>
<decorator-section-include name="body"/>
<include-screen name="footer"/>
```

We have just added a new content slot named `title` to the decorator `xhtml-decorator`.

Using Our Multi-Slot Decorator

In the file `LearningScreens.xml`, screen `CompoundedScreen` and edit the `<decorator-screen>` element's contents to add a new content section so they look like this:

```
<decorator-section name="title">
<label text="A Screen made up of 2 simpler screens."/>
</decorator-section>
<decorator-section name="body">
<include-screen name="SimplestScreen"/>
<include-screen name="AnotherSimpleScreen"/>
</decorator-section>
```

We have just added a new content section named `title`, which will be inserted into our decorator's content slot named `title`. In reality there are much simpler ways to insert a title to an HTML page. There are already a number of pre-written decorators to be found within OFBiz, these are used throughout all the application modules, giving all of the components a consistent look and feel—which can be easily changed. Take a look in `framework\common\widget\CommonScreens.xml` for examples of existing decorators.

Seeing Our First Multi-Slot Decorator

Fire an OFBiz http request `CompoundedScreen` to webapp learning and see the title changed! The title is usually displayed in the top left of the internet browser.



Nesting Decorators

It is often necessary to separate huge decorators into smaller sub-decorators, just like how we organized large screen widgets into smaller ones.

There are two approaches to nesting decorators. One is sub-decorating a sub-section, and the other is enveloping a decorator with another. It should be easier to explain these two different approaches with concrete examples, so here goes.

Top-Down Approach (delegation)

Suppose we want to decorate the XHTML document's body by giving it a light pink background. We will first need to create a wrapper (header and footer) to envelope the `<body>` element. Recall that a decorator wraps itself around the object to be decorated.

In the file `LearningScreens.xml`, create a new screen `body-header`:

```
<screen name="body-header">
<section><widgets>
    <label text="Start of Body"/>
    <label text="&lt;span style="background-color:
#FFEEEE"&gt;" />
```

```
</widgets></section>
</screen>

, a new screen body-footer:

<screen name="body-footer">
<section><widgets>
    <label text="&lt;/span&gt;"/>
    <label text="End of Body"/>
</widgets></section>
</screen>
```

and a new decorator body-decorator:

```
<screen name="body-decorator">
<section>
    <widgets>
        <include-screen name="body-header"/>
        <decorator-section-include name="body"/>
        <include-screen name="body-footer"/>
    </widgets>
</section>
</screen>
```

Note how we are using `<label>` elements in place of FreeMarker .ftl files, for the sake of brevity in example codes and convenience. All XHTML special (markup) characters, such as "<" and ">", in the `<label>` element's `text` attribute need to be escaped by using literal entities. This is alright for short text like in the above example but is inconvenient for longer text like in header1.ftl.

In the decorator `xhtml-decorator`, we are currently merely outputting the content section `body` between screens `header2` and `footer`. We now also need to pre-process the content section `body` by passing it through the decorator `body-decorator`. We do this by nesting the element `<decorator-section-include name="body"/>` in a `<decorator-screen>` and `<decorator-section>` structure like this:

```
<decorator-screen name="body-decorator">
<decorator-section name="body">
    <decorator-section-include name="body"/>
</decorator-section>
</decorator-screen>
```

Our `xhtml-decorator` should now look like:

```
<screen name="xhtml-decorator">
<section>
    <widgets>
        <include-screen name="header1"/>
        <decorator-section-include name="title"/>
```

```
<include-screen name="header2"/>
<decorator-screen name="body-decorator">
    <decorator-section name="body">
        <decorator-section-include name="body"/>
    </decorator-section>
</decorator-screen>
<include-screen name="footer"/>
</widgets>
</section>
</screen>
```

In effect, we have just told the decorator `xhtml-decorator` to delegate the decoration of the content section `body` to the sub-decorator `body-decorator`.

The result can be seen by firing an OFBiz http request `CompoundedScreen` to `webapp learning`. We will now see a light pink background plus a start and end tag showing **Start of Body** and **End of Body**, respectively.

The advantage of this approach is that each sub-decorator is kept concise and to the point. The decorator `body-decorator` clearly shows that its job is to operate on the content section `body`. The decorator shows no other content sections involved. Compare that to the decorators in the next approach.

The Bottom-Up Approach (Vertical Stack)

This approach arranges the multiple decorators into a vertical stack. The highest decorator in the stack must be the top-level decorator that deals with the highest overview of the document structure.

We first create a first level decorator named `level1-decorator` in the file `LearningScreens.xml` like this:

```
<screen name="level1-decorator">
<section>
    <widgets>
        <decorator-screen name="level2-decorator">
            <decorator-section name="title">
                <decorator-section-include name="title"/>
            </decorator-section>
            <decorator-section name="body">
                <include-screen name="body-header"/>
                <decorator-section-include name="body"/>
                <include-screen name="body-footer"/>
            </decorator-section>
        </decorator-screen>
    </widgets>
</section>
</screen>
```

We then create a higher level decorator named `level2-decorator` like this:

```
<screen name="level2-decorator">
<section>
  <widgets>
    <include-screen name="header1"/>
    <decorator-section-include name="title"/>
    <include-screen name="header2"/>
    <decorator-section-include name="body"/>
    <include-screen name="footer"/>
  </widgets>
</section>
</screen>
```

We now rewire our screen `CompoundedScreen` to use the decorator `level1-decorator` by changing the decorator screen from `xhtml-decorator`:

```
<decorator-screen name="level1-decorator">
```

Our screen `CompoundedScreen` calls on the decorator `level1-decorator`, which in turn calls on decorator `level2-decorator`, creating a vertical stack that is the bottom-up approach.

The advantage of this approach is that the top-level decorator (`level2-decorator` in this case) is kept clean and concise. It doesn't have to care about what pre-processing was done in the decorators before it. The disadvantage is that the individual sub-decorators (`level1-decorator` in this case) need to keep track of all content sections that must be eventually passed to the top-level decorator. Our sub-decorator's task should only be described as "decorate the body content", not "decorate the body content, plus do some miscellaneous house-keeping".

Using Both Approaches

Clearly, both approaches have their pros and cons. Fortunately, we can combine both approaches to get the best of both. We want to keep the top-level decorator concise, stable and not prone to change because the overall XHTML document structure is also not prone to change. We also want the flexibility to call on various sub-decorators to do special decorations on demand.

We first revert our decorator `xhtml-decorator` to the concise form it had before. We do this by replacing its `<widgets>` element's contents with this again:

```
<include-screen name="header1"/>
<decorator-section-include name="title"/>
<include-screen name="header2"/>
<decorator-section-include name="body"/>
<include-screen name="footer"/>
```

We then create an intermediate decorator that calls upon any number of special sub-decorators necessary and takes care to pass on the necessary content sections (`title` and `body`) to the top-level decorator `xhtml-decorator`. To clearly illustrate the point that this intermediate decorator is in charge of invoking any special sub-decorators, we add a new sub-decorator called `title-decorator`:

```
<screen name="title-decorator">
<section>
  <widgets>
    <label text="** Decorated Title! **: "/>
    <decorator-section-include name="body"/>
  </widgets>
</section>
</screen>
```

We now create the new intermediate decorator `intermediate-decorator` like this:

```
<screen name="intermediate-decorator">
<section>
  <widgets>
    <decorator-screen name="xhtml-decorator">
      <decorator-section name="title">
        <decorator-screen name="title-decorator">
          <decorator-section name="title">
            <decorator-section-include name="title"/>
          </decorator-section>
        </decorator-screen>
      </decorator-section>
      <decorator-section name="body">
        <decorator-screen name="body-decorator">
          <decorator-section name="body">
            <decorator-section-include name="body"/>
          </decorator-section>
        </decorator-screen>
      </decorator-section>
    </decorator-screen>
  </widgets>
</section>
</screen>
```

Finally, we rewire our screen `CompoundedScreen` to use the decorator `intermediate-decorator` by changing the decorator screen to this:

```
<decorator-screen name="intermediate-decorator">
```

Our intermediate decorator `intermediate-decorator` calls upon two sub-decorators `title-decorator` and `body-decorator`. That achieves concise and clean codes in two areas: the top-level `xhtml-decorator` which retains its clean high-level overview structure, and the individual sub-decorators which are cleanly confined to their specific duties. Thus, we have combined the two approaches, top-down and bottom-up, for the strengths of both.

To see all these in action, fire an OFBiz http request `CompoundedScreen` to `webapp learning`.

Summary

In this chapter, we have looked at:

- Equipping a webapp with a control servlet via the file `web.xml`
- Equipping a control servlet with a screen widget view handler in the file `controller.xml`
- Creating a screen widget with the `<screen>` element.
- Creating a screen widget with conditional **if-then-else structure with the `<section>` element** and its subelements `<condition>` (the `If`), `<actions>` and `<widgets>` (the `Then`), and `<fail-widgets>` (the `Else`).
- Sending parameterized requests to OFBiz with URLs that contain the "?" and "&" characters.
- Using the screen widget context in the `<actions>` element, with BeanShell script and with Minilang.
- Using nested `<section>` elements to build nested **if-then-else structures for more complex conditional screens**.
- Organizing large screen widgets into smaller nested ones with the `<include-screen>` or the `${screens.render() }` (in FreeMarker) constructs.
- Using the global context for screen widgets to share variables among nested screens.
- Creating decorator screen widgets with the `<decorator-section-include>` element for templating.
- Using decorator screen widgets with the `<decorator-screen>` and `<decorator-section>` elements.
- Nesting decorator screens with the top-down (delegation) and the bottom-up (vertical stack) approaches, and combining both approaches for the best organizational structure.

So far, we have been communicating with OFBiz webapps using requests (URLs). We've even managed to send some request parameters, in the form of GET parameters, along with those requests. Since sending GET parameters involves tagging request parameters to the request URLs, we can easily see how unwieldy that will quickly become, especially when we need to send more than a few request parameters.

In the next chapter, we will look at form widgets that will give us XHTML forms. Such forms allow us to send a great number of request parameters through a user friendly interface, without having to create mile-long request URLs.

4

Form Widgets

Form widgets are one of many types of View elements that can exist in a screen widget.

Form widgets are translated into XHTML forms by the OFBiz screen widget view handler. An XHTML form is a web-based (displayed with and operated by a web or internet browser) user-interface that allows an end-user to send a great number of request parameters to the server (OFBiz in this case). As can be seen in the last chapter's examples with GET parameters, tagging request parameters to the URL quickly becomes unmanageable with more than a few request parameters.

An OFBiz request with merely a few (three in this case) parameters can already become unwieldy: `http://localhost:8080/learning/control/SomeRequest?paramOne=valueOne¶mTwo=valueTwo¶mThree=valueThree`

It's like verbally telling a bank manager a whole string of parameters, like your full name, address, loan amount, preferred installment payment breakdowns, and so on. Better to just fill in a form!

An XHTML form lets end-users send POST parameters that are contained in the request itself, not in the request's URL.

Just as paper-based forms can have a variety of types of input boxes like text fields and checkboxes, an XHTML form also has many types of input fields

For example:

- text field (example "**Enter your name**" fields)
- checkbox (example "**Tick/Indicate your hobbies** (one or more)" fields)
- radio button (example "**Male or Female** (choose one)" fields)

and many more.

Form widgets have equivalents for most of those XHTML input fields. One of the main purposes of building XHTML forms with form widgets is to avoid the verbose and often complex syntax of XHTML forms. It should be noted though that this is a matter of personal preference. As we will learn later in the book that anything possible with Form Widgets is also possible using FreeMarker.

In this chapter, we will be looking at:

- Creating our first form widget
- Referencing form widgets from within screen widgets
- Understanding the anatomy of an XHTML form
- Minimal requirements to use form widgets
- Form processing via request events
- Preparing the data for the form widget using BeanShell
- Handling and processing the action using Java
- Type "list" form widgets
- Type "multi" form widgets
- When to use form widgets.

Files and Locations

Form widgets are usually defined in an OFBiz component's widget folder and the same goes for screen widgets. The filename convention in OFBiz for files containing screen widgets is a string that ends with `Forms.xml`.

Creating Our First Form Widget

A form widget cannot exist outside of a screen widget. We will first need to create a screen widget that contains our first form widget. This will be referred to as the containing screen widget.

Creating the Containing Screen Widget

We first create a screen widget in which we can insert a form widget. In the file `LearningScreens.xml`, create a new screen `OneFormScreen` like this:

```
<screen name="OneFormScreen">
<section>
  <widgets>
    <decorator-screen name="xhtml-decorator">
      <decorator-section name="title">
```

```
<label text="A Screen with just 1 form."/>
</decorator-section>
<decorator-section name="body">
    <include-form name="FirstForm"
        location="component://learning/widget/
            learning/LearningForms.xml"/>
    </decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>
```

Edit the file \${webapp:learning}\WEB-INF\controller.xml, and add a new request map like this:

```
<request-map uri="OneForm">
    <response name="success" type="view" value="OneFormScreen"/>
</request-map>
```

and a new view map like this:

```
<view-map name="OneFormScreen" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#OneFormScreen"/>
```

Referencing Form Widgets

The `<include-form>` element includes a form named `FirstForm` residing in a file `LearningForms.xml`. In this book, referencing form widgets is the same as referencing screen widgets. For example, the above form widget will be referred to in this book as form `FirstForm` in `component://learning/widget/learning/LearningForms.xml`.

In the code, however, the `<include-form>` element is used in such a way that the form name and the form location (file in which form resides) is always separate; form widgets reside in `<forms>` elements, not `<screens>` elements like screen widgets do. Therefore, the `location` attribute is always required.

Create the Form Widget

A form widget is defined by a `<form>` element. A form is made up of fields, which can hold individual units of data or information to be sent to the server as request parameters. It can also contain one or more **Submit** buttons, which submit the entire form to the server when clicked.

We now create a form widget named `FirstForm`. In the folder `${component:learning}\widget\learning`, create a new file `LearningForms.xml`, and enter into it this:

```
<forms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/
                                         dtds/widget-form.xsd">
<form name="FirstForm" type="single" target="processFirstForm">
<field name="firstName"><text/></field>
<field name="lastName"><text/></field>
<field name="submit"><submit/></field>
</form>
</forms>
```

Seeing Our First Form

Fire an OFBiz http request `OneForm` to webapp `learning`. We see two fields **First Name** and **Last Name**, and one **Submit** button.

The image shows a simple web form enclosed in a light gray border. It contains two text input fields stacked vertically. The top field is labeled "First Name" and the bottom field is labeled "Last Name". Below these fields is a single "Submit" button.

Note how the Widget Engine automatically generates a field's label **First Name** from the field's name `firstName`. The Widget Engine generates the field's label from the field's name written in **Camel Case** (such as `firstName`) such that a name of `longFieldName` will produce a label of **Long Field Name**. This is just one of the many neat features in the Widget Engine.

Understanding the Form Attributes

View the page source to see a line like this:

```
<form method="post" action="/learning/control/processFirstForm"
      class="basic-form"
      onSubmit="javascript:submitFormDisableSubmits(this)"
      name="FirstForm">
```

The `method` attribute specifies the method of submission of the form. It can be `POST` or `GET` with `POST` being more secure. OFBiz will always create web forms with the `POST` method. This is the attribute that determines whether request parameters sent to the server are `GET` parameters or `POST` parameters. As can be seen, there are two ways to send `GET` parameters: via parameterized URLs or with `GET` forms. The only way to send `POST` parameters is with `POST` forms.

The `action` attribute specifies the request to submit the form to. Note how OFBiz's form widget renderer automatically generates an OFBiz request string to `webapp learning`; the form widget's `target` attribute has nothing more than `processFirstForm`. The framework will construct an OFBiz request to the same `webapp` that the form widget is called from, in this case `learning`. The form widget `FirstForm` is called from the screen widget `OneFormScreen`, which is called from the view map `OneFormScreen` in the controller.xml file of `webapp learning`.

The `name` attribute uniquely identifies the web form on the page. On the XHTML page and in that context, this unique identifier is usually referenced in JavaScript, something we won't be covering in this book. In the context of writing form widgets, we just need to know that the form name allows us to refer to it from a screen widget with the `<include-form>` element.

The `class` attribute specifies a particular style with which to present this form. Styles could mean to display with gray background or to add frilly borders or any other creative ways to present the form. The technology involved here is **Cascading Style Sheets (CSS)**. Style sheets describe the presentation of structured documents; XHTML documents are structured. Hence, `basic-form` is simply the name of a style. We notice that this style is not defined in our page. We will need to include this style's definition from somewhere. But before we correct this, there is something else missing that we need to see to.

The `onSubmit` attribute is one of many JavaScript events that a web form can encounter. These events are called DOM events, where DOM stands for **Document Object Model**. This particular event `onSubmit` occurs when the form is submitted. The value of the attribute `onSubmit` specifies the JavaScript function to execute when the event is encountered. This function is missing from our page, so we will need to include it too.

Minimum Requirements to Use Form Widgets

On rendering our first form widget, we see two things missing: the CSS style `basic-form` and the JavaScript function `submitFormDisableSubmits`. We now look for these missing pieces.

The CSS StyleSheet `maincss.css`

OFBiz uses a single CSS style sheet file `maincss.css` residing in component `images` which is located in `${OFBizInstallFolder}\framework\images`. Specifically, the file is in the folder `${component:images}\webapp\images`, or rather in `${webapp:images}` because that folder is a `webapp` in component `images`. It follows that the request needed to access this file is `images\maincss.css`.

The JavaScript Library `selectall.js`

The required JavaScript function `submitFormDisableSubmits` resides in a JavaScript library file `selectall.js` in the folder `${webapp:images}`. Since it is in the same location as the file `maincss.css`, the request to access it is `images\selectall.js`.

The JavaScript function `submitFormDisableSubmits` is used in every form widget.

Including the Minimal Requirements

We need to include the style sheet and the JavaScript library in our page. Since our page and those required files all reside on the same server location `http://localhost:8080`, we can omit the server location when accessing those files. Edit the file `${webapp:learning}\includes\header1.ftl`, and insert between the tags `<head>` and `<title>` this:

```
<script language="javascript" src="/images/selectall.js"
    type="text/javascript"></script>
<link rel="stylesheet" href="/images/maincss.css" type="text/css"/>
```

Form Processing via Request Event

A request event is a part of a request map. All processing, especially data-modifying processing, are filed under request events. A request event is declared by an `<event>` sub-element, and defined by a processing script written in one of a number of possible languages such as Java, BeanShell, or Minilang.

The full anatomy of the `<request-map>` element will be covered later in the book. For now, we only need to know that the sub-element `<event>` specifies the processing logic to use to process the incoming request. Each request map can have one request event.

To be able to process the form, we first need to tell our webapp's control servlet to listen to our form submission. Our form's action attribute is `processFirstForm`, so we need to add a request map named `processFirstForm` to our webapp's `controller.xml`. Edit `${webapp:learning}\WEB-INF\controller.xml` and add the following request map:

```
<request-map uri="processFirstForm">
    <event type="java" path="org.ofbiz.learning.learning.LearningEvents"
        invoke="processFirstForm"/>
    <response name="success" type="view" value="OneFormScreen"/>
</request-map>
```

Still in the controller add the Java Event Handler, so the control servlet knows how to handle this type of event. At the top of the file, immediately above the screen handler, add:

```
<handler name="java" type="request" class="org.ofbiz.webapp.event.JavaEventHandler"/>
```

The logic resides in a method called `processFirstForm` in the class `org.ofbiz.learning.learning.LearningEvents` we created in Chapter 2.

```
public static String processFirstForm(HttpServletRequest request,
    HttpServletResponse response) {
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    request.setAttribute("combined", firstName + " " + lastName);
    request.setAttribute("allParams", UtilHttp.getParameterMap(request));
    request.setAttribute("submit", "Submitted");
    return "success";
}
```

Stop OFBiz if it is running and compile just the Learning component by double clicking the **OFBiz - Learning Component** entry in the **Ant** window in Eclipse. You may get a **BUILD FAILED** message, in which case you must add the import statement for the `UtilHttp` class, a handy way to do this is to place the cursor immediately after the unrecognized class name in the Code Editor window and press `Ctrl + Space`. Once this is done, rebuild and then restart OFBiz.

Java Events

The event handler for Java (Java class `JavaEventHandler`) invokes the method stated in the `invoke` attribute of the `<event>` element with the parameters `request` and `response`. The `ControlServlet` class has already added a number of attributes to the `request` and these can be retrieved from within the method and used as we need them.

The `locale` object can be retrieved like this:

```
Locale locale = UtilHttp.getLocale(request);
```

The `delegator` object can be retrieved like this:

```
GenericDelegator delegator =
    (GenericDelegator) request.getAttribute("delegator");
```

The `dispatcher` object can be retrieved like this:

```
LocalDispatcher dispatcher =
    (LocalDispatcher) request.getAttribute("dispatcher");
```

The security object can be retrieved like this:

```
Security security = (Security)request.getAttribute("security");
```

The userLogin object can be retrieved like this:

```
GenericValue userLogin =
(GenericValue)request.getSession().getAttribute("userLogin");
```

Submitting and Processing Our First Form

Submitting our first form is simple enough. Don't do this yet, but it's as simple as clicking the **Submit** button.

We have just sent into the request (Java object of type `HttpServletRequest`) the new attributes `combined` and `allParams`. The value of `combined` is a combination or concatenation of the fields `firstName` and `lastName`; `allParams` is a verbose list of all parameters received by the script. We need to display this processing to the end-user. We insert the code for such a display in the final page presented to the end-user. The flow goes from screen widget `OneFormScreen` (invoked via `request / OneForm`) to BeanShell script `processFirstForm.bsh` and then back to screen widget `OneFormScreen`. We insert our code in the screen widget `OneFormScreen`. Right below this:

```
<include-form name="FirstForm"
    location="component://learning/widget/learning/
    LearningForms.xml"/>
```

insert this:

```
<section>
<condition><not><if-empty field-name="parameters.submit"/>
</not></condition>
<widgets>
<container><label text="firstName: ${parameters.firstName}"/>
</container>
<container><label text="lastName: ${parameters.lastName}"/>
</container>
<container><label text="combined: ${parameters combined}"/>
</container>
<container><label text="submit: ${parameters.submit}"/></container>
<container><label text="All the parameters we received:"/>
</container>
<container><label text="${parameters.allParams}"/></container>
</widgets>
</section>
```

Note how we are using a nested `<section>` element to create a nested conditional. The top-level condition for the screen widget `OneFormScreen` is to display under all circumstances (that is, zero conditions). The form portion of the screen, which is the `<include-form>` element, is always displayed. A sub-portion of the screen, the results of processing the form, is displayed only when the form is submitted. The "flag" that indicates whether the form is submitted is in the request attribute `submit`, which we tagged on whenever our script `processFirstForm` is executed.

Note also that the parameters `combined` and `allParams` are request attributes, not request parameters. See the use of the `request.setAttribute` method in our `processFirstForm` method. Recall that a screen widget's `parameters` variable is composed of request attributes too. That is why `${parameters.combined}` and `${parameters.allParams}` work.

Test the form, submitting and processing it. Note the parameters listed under **All the parameters we received:**

The "list" Type Form Widget

In the previous section, we created a single type form widget. Here, we look at the `list` type form widget. The `list` type form widget is good for displaying a list of similar data structures for the end-user to edit. It is like displaying a list of different types of loan application forms, with all forms having the same structure. Some hands-on experience will better explain this.

Creating the Containing Screen

Every widget must be contained in a screen (unless it is itself a screen). We create the containing screen widget here.

In file `LearningScreens.xml`, create a screen widget `ListFormScreen` like this:

```
<screen name="ListFormScreen">
<section>
<actions>
    <set field="row1.firstName" value="First1"/>
    <set field="row1.lastName" value="Last1"/>
    <set field="row2.firstName" value="First2"/>
    <set field="row2.lastName" value="Last2"/>
    <set field="row3.firstName" value="First3"/>
    <set field="row3.lastName" value="Last3"/>
    <set field="listOfRecords []" from-field="row1"/>
    <set field="listOfRecords []" from-field="row2"/>
    <set field="listOfRecords []" from-field="row3"/>
</actions>
```

```
<widgets>
  <decorator-screen name="xhtml-decorator">
    <decorator-section name="title">
      <label text="A Screen with a list of forms."/>
    </decorator-section>
    <decorator-section name="body">
      <include-form name="ListForm"
                    location="component://learning/widget/learning/
                               LearningForms.xml"/>
      <include-screen name="ShowProcessing"/>
    </decorator-section>
  </decorator-screen>
</widgets>
</section>
</screen>
```

In the `<actions>` element, we created a list of three values. Such values in a list are generally called **rows** or **records**, because we will often pull in a list of similarly structured records like say "a list of postal addresses" from the database. While the `single` type form widget only shows a single record, the `list` type form widget can show a list of records.

Each record is a Map of two fields named `firstName` and `lastName`. This corresponds to how a database record is structured, with a group of fields put together to form a collection of keys (field names such as `firstName` and `lastName`) and values (field values such as `OFBiz` and `Researcher`).

Adding Form Processing Code

Note that we are now using `<include-screen name="ShowProcessing"/>` to display the processing. The same codes used in the screen widget `OneFormScreen` will also be used here, so it now makes sense to refactor that chunk of code into a reusable screen `ShowProcessing` like this:

```
<screen name="ShowProcessing">
<section>
  <condition><not><if-empty field-name="parameters.submit"/>
  </not></condition>
<widgets>
  <container><label text="firstName: ${parameters.firstName}" />
  </container>
  <container><label text="lastName: ${parameters.lastName}" />
  </container>
  <container><label text="combined: ${parameters.combined}" />
  </container>
  <container><label text="submit: ${parameters.submit}" />
  </container>
```

```
<container><label text="All the parameters we received:"/>
</container>
<container><label text="${parameters.allParams}" /></container>
</widgets>
</section>
</screen>
```

To complete the refactor, get screen widget OneFormScreen to use screen ShowProcessing as well.

Publishing the Form

Publish the screen ListFormScreen by adding to \${webapp:learning}\WEB-INF\controller.xml this request map:

```
<request-map uri="ListForm">
<response name="success" type="view" value="ListFormScreen"/>
</request-map>
```

and this view map:

```
<view-map name="ListFormScreen" type="screen"
page="component://learning/widget/learning/
LearningScreens.xml#ListFormScreen"/>
```

In the file LearningForms.xml, add a new form widget ListForm:

```
<form name="ListForm" type="list" list-name="listOfRecords"
target="processListForm" separate-columns="true">
<field name="firstName"><text/></field>
<field name="lastName"><text/></field>
<field name="submit"><submit/></field>
</form>
```

The form takes a list of values listOfRecords and displays a single web form for each record in the list.

The form's target action, the "instruction" or "request" to process the form, is processListForm. In the file controller.xml, add another request map processListForm:

```
<request-map uri="processListForm">
<event type="java" path="org.ofbiz.learning.learning.LearningEvents"
invoke="processFirstForm"/>
<response name="success" type="view" value="ListFormScreen"/>
</request-map>
```

Seeing Our First "list" Type Form

Fire an OFBiz http request `ListForm` to webapp learning, and test each **Submit**. There is no need to recompile since we have not changed any of the Java code.

First Name	Last Name	Submit
First1	Last1	Submit
First2	Last2	Submit
First3	Last3	Submit

Notice that there really are three separate web forms, each with one **Submit** button. When a **Submit** button is clicked, its corresponding web form is submitted. The `processFirstForm` method in `LearningEvents.java` is used because that script is made to handle only one web form at a time, which is perfect for this case.

Test the "list" form to confirm that there are indeed three separate web forms on the page. Also confirm that each **Submit** button will submit only a single form with three parameters: `firstName`, `lastName`, and `submit`, which can be seen in the output under the text **All the parameters we received:**

For example, submitting the second form will show:

```
firstName: First2
lastName: Last2
combined: First2 Last2
submit: Submitted
All the parameters we received:
[lastName=Last2, firstName=First2, submit=Submit]
```

The "multi" Type Form Widget

It is often useful to edit a whole list of records in one go, and then click a single **Submit** button to update them all. The `single` and the `list` type form widgets cannot allow this; they can only allow the update of a single record at every click of a **Submit** button. We now look at the `multi` type form widget.

Creating the Containing Screen

In the file `LearningScreens.xml`, create a screen widget `MultiFormScreen` like this:

```
<screen name="MultiFormScreen">
<section>
  <actions>
```

```

<script location="component://learning/webapp/learning/
WEB-INF/actions/learning/loadListOfSampleData.bsh"/>
</actions>
<widgets>
    <decorator-screen name="xhtml-decorator">
        <decorator-section name="title">
            <label text="A Screen with a multi type form."/>
        </decorator-section>
        <decorator-section name="body">
            <include-form name="MultiForm"
                location="component://learning/widget/
learning/LearningForms.xml"/>
            <include-screen name="ShowProcessing"/>
        </decorator-section>
    </decorator-screen>
</widgets>
</section>
</screen>

```

Loading Data for the Form

Note that we are now using `<script>` to load the list of sample data. The same script actions used in the screen widget `ListFormScreen` will also be used here, so it now makes sense to refactor that chunk of script actions into a consolidated BeanShell script `loadListOfSampleData.bsh` like this:

```

// java.util.* (HashMap and ArrayList) imported by default
ArrayList listOfRecords = new ArrayList();
for(int i=1; i<=3; i++){
    HashMap row = new HashMap();
    row.put("firstName", "First" + i);
    row.put("lastName", "Last" + i);
    listOfRecords.add(row);
}
context.put("listOfRecords", listOfRecords);

```

Note the `context` variable. This doesn't exist when we call a BeanShell script from a request map, like in the script `processListForm` when it is called by the request map `processListForm`. It only exists when we call a BeanShell script from the pre-processing portion of a screen widget, which is encapsulated in the `<actions>` element.

The context variable represents the environment or context of a screen widget where variables can be stored, updated, and used. One example of such variables in the context is the `parameters` variable. Variables stored in the screen widget's context can be referenced in the screen widget's codes.

Complete the refactor by getting the screen widget `ListFormScreen` to use this BeanShell script as well.

Publishing the Form

Publish the screen `MultiFormScreen` by adding to `${webapp:learning}\WEB-INF\controller.xml` this request map:

```
<request-map uri="MultiForm">
    <response name="success" type="view" value="MultiFormScreen"/>
</request-map>
```

and this view map:

```
<view-map name="MultiFormScreen" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#MultiFormScreen"/>
```

In the file `LearningForms.xml`, add a new form widget `MultiForm`:

```
<form name="MultiForm" type="multi" list-name="listOfRecords"
    target="processMultiForm" separate-columns="true">
    <field name="firstName"><text/></field>
    <field name="lastName"><text/></field>
    <field name="_rowSubmit" title="Select"><check/></field>
    <field name="submit"><submit/></field>
</form>
```

Like the `list` type form widget, the `multi` type form widget takes a list of values `listOfRecords`. However, instead of displaying a single web form for each record, the `multi` type form widget displays just one web form and one **Submit** button for all the records. This allows the end-user to update all the records at once with a single click of the single **Submit** button.

The form's target action is `processMultiForm`. In the file `controller.xml`, add a request map `processMultiForm`:

```
<request-map uri="processMultiForm">
    <event type="java" path="org.ofbiz.learning.learning.LearningEvents"
        invoke="processMultiForm"/>
    <response name="success" type="view" value="MultiFormScreen"/>
</request-map>
```

Creating the Form-Processing Logic

Create the processing logic in a new static method `processMultiForm` in our `LearningEvents` class, and enter into it:

```
public static String processMultiForm (HttpServletRequest request,
    HttpServletResponse response) {
    Collection parsed = UtilHttp.parseMultiFormData
        (UtilHttp.getParameterMap(request));
    List combined = new ArrayList();
    Iterator parsedItr = parsed.iterator();
    while (parsedItr.hasNext()) {
        Map record = (Map)parsedItr.next();
        combined.add(record.get("firstName") + " " +
            record.get("lastName"));
    }
    request.setAttribute("combined", combined);
    request.setAttribute("allParams",
        UtilHttp.getParameterMap(request));
    request.setAttribute("submit", "Submitted");
    return "success";
}
```

The `multi` type form widget submits the field values with special suffixes attached to the field names. Since the list of records contain similar structures such that every record has the same fields (`firstName` and `lastName`), combining them all into a single web form to be submitted all at once requires that we be able to distinguish between one record's `firstName` and another's. This is done by appending each field name with `_o_<n>`, where `<n>` is a number that indicates the record's position in the list. In our case, it is between **zero** to **two** because we have three records in our list of sample data.

The special function `UtilHttp.parseMultiFormData` takes all those fields and puts them neatly into a list of records again. For now all we need to know is that the `UtilHttp.parseMultiFormData` method is indispensable when using `multi` type form widgets.

Seeing Our First "multi" Type Form

If OFBiz is running, shut it down; rebuild the Learning component and restart, and then fire an OFBiz http request `MultiForm` to webapp `learning`. There is a checkbox beside every record. A record will only be processed if its checkbox is checked. Technically speaking though, all records are submitted but only the selected ones will be processed.

For example, checking the first and third record, and then submitting, will produce the following output:

First Name	Last Name	Select
First1	Last1	<input type="checkbox"/>
First2	Last2	<input type="checkbox"/>
First3	Last3	<input checked="" type="checkbox"/>

Submit

```
firstName:  
lastName:  
combined: [First3 Last3, First1 Last1]  
submit: Submitted  
All the parameters we received:  
[firstName_o_2=First3, firstName_o_1=First2, firstName_o_0=First1,  
lastName_o_2=Last3, lastName_o_1=Last2, lastName_o_0=Last1,  
submit=Submit, _rowSubmit_o_2=Y, _rowSubmit_o_0=Y, _rowCount=3]
```

Alternative Targets in Two-Target Forms

The `<alt-target>` element allows a form to have an alternative action attribute. Recall that the `action` attribute of an XHTML form is the request via which the form is submitted, and that all communications with a web application (like OFBiz) are done through requests.

This `<alt-target>` is often (and possibly only) used when attempting to reuse a single form for both the "update existing record" and the "create new record" scenarios.

Creating the Form

In the file `LearningForms.xml`, add a new form widget `TwoTargetForm`:

```
<form name="TwoTargetForm" type="single" target="updateRecord"  
      default-map-name="record">  
    <alt-target target="createRecord" use-when="record==null"/>  
    <field name="firstName"><text/></field>  
    <field name="lastName"><text/></field>  
    <field name="submit"><submit/></field>  
</form>
```

This form has a default `action` attribute of `updateRecord`, and an alternative of `createRecord`. Both target requests are non-existent; we just want to see how `<alt-target>` works.

Creating the Containing Screen

In the file `LearningScreens.xml`, add a new form widget `TwoTargetFormScreen`:

```
<screen name="TwoTargetFormScreen">
    <section>
        <actions>
            <entity-one entity-name="Person" value-name="record"/>
        </actions>
        <widgets>
            <decorator-screen name="xhtml-decorator">
                <decorator-section name="title">
                    <label text="A Screen with a 2-target form."/>
                </decorator-section>
                <decorator-section name="body">
                    <include-form name="TwoTargetForm"
                        location="component://learning/widget/
                        learning/LearningForms.xml"/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

The screen uses `<entity-one>` to retrieve a record from the database, a record of type `Person`. The `<entity-one>` element will be explored in further detail when we cover entities later in the book. For now, we just need to know that variable `record` will be null if no record can be retrieved.

Publishing the Two-Target Form

Publish the screen `TwoTargetFormScreen` by adding to `${webapp:learning}\WEB-INF\controller.xml` this request map:

```
<request-map uri="TwoTargetForm">
    <response name="success" type="view" value="TwoTargetFormScreen"/>
</request-map>
```

and this view map:

```
<view-map name="TwoTargetFormScreen" type="screen"
    page="component://learning/widget/learning
    /LearningScreens.xml#TwoTargetFormScreen"/>
```

Seeing Our First Two-Target Form

The `<entity-one>` element expects a primary key (explained later when covering entities). The primary key for records of type **Person** is **partyId**.

Fire an http OFBiz request `TwoTargetForm` to `webapp learning`, sending along a GET parameter `partyId` of value `admin`. We should see the two-target form with two of the record's fields:

First Name	<input type="text" value="THE"/>
Last Name	<input type="text" value="ADMINISTRATOR"/>
Submit	

Don't submit the form! Both of the form's targets are non-existent.

The page source should show this:

```
<form method="post" action="/learning/control/updateRecord" ...>
```

Since the screen's `<entity-one>` element was able to retrieve an existent database record, it generates the form for the **update existing record** scenario.

Omitting the GET parameter `partyId`, or setting it to a value that doesn't exist in the database (example "blahblah") will generate a form with a different action attribute:

```
<form method="post" action="/learning/control/createRecord" ...>
```

Since the screen's `<entity-one>` element was unable to retrieve an existent database record, it generates the form for the **create new record** scenario.

Row-Level Actions

Form widgets can have a child element `<actions>`, just like screen widgets. In addition, form widgets can also have a child element `<row-actions>`. There is nothing like this for screen widgets, so this child element warrants some explanation here.

The `<row-actions>` element is used in forms that deal with lists of record, that is the `list` and `multi` type forms. The actions specified inside the `<row-actions>` element will be executed for each record in the list.

Creating the Form

In the file `LearningForms.xml`, add a new form widget `RowMutatingForm`:

```
<form name="RowMutatingForm" type="list" list-name="listOfRecords"
      target="processListForm"
      separate-columns="true">
  <row-actions>
    <set field="firstName" value="${firstName} Name"/>
    <set field="lastName" value="${lastName} Name"/>
  </row-actions>
  <field name="firstName"><text/></field>
  <field name="lastName"><text/></field>
  <field name="submit"><submit/></field>
</form>
```

This form is really a copy of the `ListForm`, but with the `<row-actions>` added.

The actions in the `<row-actions>` element serve to tag on the text `Name` behind every row's field values (values of fields `firstName` and `lastName`).

Creating the Containing Screen

In the file `LearningScreens.xml`, add a new form widget `RowMutatingFormScreen`:

```
<screen name="RowMutatingFormScreen">
  <section>
    <actions>
      <set field="row1.firstName" value="First1"/>
      <set field="row1.lastName" value="Last1"/>
      <set field="row2.firstName" value="First2"/>
      <set field="row2.lastName" value="Last2"/>
      <set field="row3.firstName" value="First3"/>
      <set field="row3.lastName" value="Last3"/>
      <set field="listOfRecords []" from-field="row1"/>
      <set field="listOfRecords []" from-field="row2"/>
      <set field="listOfRecords []" from-field="row3"/>
    </actions>
    <widgets>
      <decorator-screen name="xhtml-decorator">
        <decorator-section name="title">
          <label text="A Screen with a list of forms." />
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

```
<decorator-section name="body">
    <include-form name="RowMutatingForm"
                  location="component://learning/widget/
                             learning/LearningForms.xml"/>
    <include-screen name="ShowProcessing"/>
</decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>
```

This screen is a copy of the `ListFormScreen`, except that it includes the form `RowMutatingForm` instead of the `ListForm`.

Publishing the Form

Publish the screen `RowMutatingFormScreen` by adding to `${webapp:learning}\WEB-INF\controller.xml` this request map:

```
<request-map uri="RowMutatingForm">
    <response name="success" type="view"
              value="RowMutatingFormScreen"/>
</request-map>
```

and this view map:

```
<view-map name="RowMutatingFormScreen" type="screen"
          page="component://learning/widget/learning/
                           LearningScreens.xml#RowMutatingFormScreen"/>
```

Seeing the Form in Action

Fire an http OFBiz request `RowMutatingForm` to webapp `learning`. We should see the familiar `ListForm` form, but with fields of all records mutated:

First Name	Last Name	Submit
First1 Name	Last1 Name	Submit
First2 Name	Last2 Name	Submit
First3 Name	Last3 Name	Submit

Summary

In this chapter, we looked at:

- Creating our first form widget within a `<forms>` element in file `LearningForms.xml`.
- Referencing form widgets from within screen widgets by using the `<include-form>` element.
- Understanding the anatomy of an XHTML form, parts such as `method`, `action`, `name`, `class` and `onSubmit`, so that we know how a form can be referenced and processed.
- Minimal requirements to use form widgets, which are the style sheet `maincss.css` and the JavaScript library `selectall.js`.
- Form processing via request events, which are declared in `controller.xml`.
- Preparing data for a form using BeanShell
- Programming the request-event-handling using Java.
- Type `list` form widgets, which really are multiple forms arranged for easy viewing at a glance.
- Type `multi` form widgets, which really is a single form with a list of values (records) that can be updated all at once by clicking on the single **Submit** button for the single form.

In the next chapter, we will look at other useful types of View elements that can exist in a screen widget. Form widgets are a major component of screens, but there are other crucial components as well.

5

Other View Element Types in Screen Widgets

After the last two weighty chapters on the biggest components of OFBiz widgets, screen and form, we now look at other widget types that are nonetheless crucial.

We will first look at menu widgets. An application cannot live without menus. users need menus to navigate an application.

We then look at OFBiz widget's integration with FreeMarker, a templating engine that generates XHTML code. OFBiz widgets themselves cannot intermingle directly with XHTML code—that is to say, an OFBiz widget will not work if it is placed directly inside the template engine code. Although this insulation makes for clean code that describes OFBiz widgets neatly, there is much XHTML that cannot be produced using OFBiz widgets.

Although OFBiz widgets are still a work in progress, despite being already full-fledged in several vital areas, it is unlikely that OFBiz widgets will be further extended to reinvent everything good in XHTML. OFBiz widgets already do cover a huge and commonly useful majority of XHTML constructs.

Still, there are parts of XHTML that cannot be easily translated into a simpler structure that is OFBiz widgets, especially parts that describe complex user-interfaces.

As such, we combine OFBiz widgets with FreeMarker, gaining the powerful ability to leverage on the simplicity of OFBiz widgets (80-90 percent of the time) as well as the vast creativity possible with XHTML.

In this chapter, we will be looking at:

- Menu widgets
- Creating our first menu widget

- Embedding or including menu widgets into screen widgets
- Creating sub-menus with menu widgets
- Preprocessing for menu widgets, much like the `<actions>` element for screen widgets.
- OFBiz's widget (View component) integration with FreeMarker
- Using FreeMarker files as decorator templates
- Displaying dynamically created list variables
- Re-using existing screen widgets

Menu Widgets

Menu widgets are a type of View element inside a screen widget. They provide a convenient means to display a menu with menu items.

Thus, so far, our previous examples and experiments have not been organized into a neat end-user menu. Let us do that now.

Creating Our First Menu Widget

In the folder `${component:learning}\widget\learning`, create a file `LearningMenus.xml` and enter into it this:

```
<?xml version="1.0" encoding="UTF-8"?>
<menus xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation=
           "http://www.ofbiz.org/dtds/widget-menu.xsd">
    <menu name="TopMenu">
    </menu>
</menus>
```

We have just created our first menu widget, but there are no menu items in the menu yet. We'll add some menu items soon. For now, we need to create a common decorator screen widget to include this menu into our screen widgets. In the file `LearningScreens.xml`, create a screen widget `CommonLearningDecorator`:

```
<screen name="CommonLearningDecorator">
    <section>
        <widgets>
            <decorator-screen name="xhtml-decorator">
                <decorator-section name="title">
                    <decorator-section-include name="title"/>
                </decorator-section>
                <decorator-section name="body">
                    <include-menu name="TopMenu"
```

```
location="component://learning/widget/
           learning/LearningMenus.xml"/>
<decorator-section-include name="body"/>
</decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>
```

Including Our Menu Widget in Our Screen Widgets

Let us add our first screen widget `SimplestScreen` as the first menu item in our menu widget `TopMenu`. In the menu widget `TopMenu`, add the following menu item:

```
<menu-item name="SimplestScreen" title="Simplest Screen">
  <link target="SimplestScreen"/>
</menu-item>
```

Understanding the Menu Item Attributes

The `name` attribute is important. It is used to identify the menu item. A menu item can be displayed as "selected" or not. If displayed as "selected", it indicates to the end-user that the page being viewed falls under this menu item.

The `title` attribute contains the text to show for the menu item.

The sub-element `<link>` indicates the request to send to the webapp when the menu item is clicked. By default, the `<link>` element's attribute `url-mode` (not specified in example) is `intra-app`, which assumes the request path (`SimplestScreen` in this case) should be tagged on to the webapp's request path (`/learning/control/` in this case). More details on this will be explained later in the book.

Including the Menu Widget via a Decorator Screen Widget

Now we must include our menu widget in our first screen widget `SimplestScreen`. In the file `LearningScreens.xml`, create a new screen widget `SimplestScreenWithMenu` to wrap the decorator `CommonLearningDecorator` around the screen widget `SimplestScreen`:

```
<screen name="SimplestScreenWithMenu">
  <section>
    <actions><set field="tabButtonItem"
           value="SimplestScreen"/></actions>
```

```
<widgets>
  <decorator-screen name="CommonLearningDecorator">
    <decorator-section name="body">
      <include-screen name="SimplestScreen"/>
    </decorator-section>
  </decorator-screen>
</widgets>
</section>
</screen>
```

and change the view map SimplestScreen in the file controller.xml to this:

```
<view-map name="SimplestScreen" type="screen"
  page="component://learning/widget/
  learning/LearningScreens.xml#SimplestScreenWithMenu"/>
```

In the `<actions>` element, we had assigned to a variable `tabButtonItem` a value of `SimplestScreen`. The value is the name of the menu item we just created above. This tells the menu widget to highlight the menu item named `SimplestScreen`.

The default variable name that the menu widget looks out for is `tabButtonItem`. This can be overridden by specifying in the menu widget (`<menu>` element) the attribute `selected-menuitem-context-field-name`.

Fire an OFBiz http request `SimplestScreen` to webapp `learning` to see the menu widget in action. At this point, there is only one menu item, so it's not easy to see that it is indeed highlighted. Let us add another menu item.

Add Screen Widget "ConditionalScreen" to the Menu

In the menu widget `TopMenu`, add the following menu item:

```
<menu-item name="ConditionalScreen" title="Conditional Screen">
  <link target="ConditionalScreen"/>
</menu-item>
```

In the file `LearningScreens.xml`, create a new screen widget

`ConditionalScreenWithMenu` to wrap the decorator `CommonLearningDecorator` around the screen widget `ConditionalScreen`:

```
<screen name="ConditionalScreenWithMenu">
  <section>
    <actions><set field="tabButtonItem"
      value="ConditionalScreen"/></actions>
  <widgets>
    <decorator-screen name="CommonLearningDecorator">
      <decorator-section name="body">
```

```

<include-screen name="ConditionalScreen" />
</decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>

```

and finally we change the view map ConditionalScreenWithMenu in the file controller.xml to this:

```

<view-map name="ConditionalScreen" type="screen"
page="component://learning/widget/learning/
LearningScreens.xml#ConditionalScreenWithMenu"/>

```

This example relies on the screen widget ConditionalScreen which was created at the beginning of Chapter 3.

Return to the browser window and refresh. We now have two menu items to choose from. Click on them to see the difference between highlighted and non-highlighted menu items.

Clicking on menu item **Simplest Screen** will show the screen SimplestScreen, highlight the menu item **Simplest Screen**, and show the other menu items (**Conditional Screen** only, for now) as non-highlighted:



The screen widget ConditionalScreen accepts a request parameter show to display two different scenarios. We will look at sub-menus next.

Sub-Menus and Conditional-Menu Items

Menus can be arranged in any number of levels. Let us now look at creating our first sub-menu. First, we change the screen widget ConditionalScreenWithMenu to include a new menu as a sub-menu. In the screen widget ConditionalScreenWithMenu, insert into the <decorator-screen> element an <include-menu> element like this:

```

<decorator-screen name="CommonLearningDecorator">
  <decorator-section name="body">
    <include-menu name="ConditionalScreenSubMenu"
      location="component://learning/widget/
      learning/LearningMenus.xml"/>

```

That is how we include our sub-menu `ConditionalScreenSubMenu`, it is as simple as that. The decorator screen `CommonLearningDecorator` displays the top-level menu. The sub-menu included inside the `<decorator-screen>` element will be displayed below the top-level menu.

We now create the sub-menu. In the file `LearningMenus.xml`, create a new menu widget `ConditionalScreenMenu`:

```
<menu name="ConditionalScreenSubMenu">
    <menu-item name="showWidgets" title="Show &lt;widgets&gt;">
        <condition>
            <if-compare field-name="parameters.show" operator="not-equals"
                value="widgets"/>
        </condition>
        <link target="ConditionalScreen?show=widgets"/>
    </menu-item>
    <menu-item name="showFailWidgets"
        title="Show &lt;fail-widgets&gt;">
        <condition>
            <if-compare field-name="parameters.show" operator="equals"
                value="widgets"/>
        </condition>
        <link target="ConditionalScreen"/>
    </menu-item>
</menu>
```

We have two scenarios, to show the `<widgets>` element or the `<fail-widgets>`. We put the two scenarios under two menu items.

Note how we used a `<condition>` element inside each `<menu-item>` element to conditionally show those menu items. If the condition fails, the menu item is not shown. In this case, we show the menu item to activate the scenario not in effect at the moment. If the scenario in effect is where the `<widgets>` element is shown, we show the menu item to activate the `<fail-widgets>` element. And vice versa if the other scenario is in effect.

In the above example, the attribute `name` of the `<menu-item>` elements don't matter because they are not referenced anywhere. We cheated by putting the two possible scenarios under the display of a single menu item that serves as a toggle.

When the `<widgets>` element is shown, the **Show <fail-widgets>** menu item is presented, allowing the user the view the alternative:



Conversely, when the `<fail-widgets>` element is shown, the **Show <widgets>** menu item is presented instead:



Pre-processing Actions for Menu Widgets

The `<menu>` element can contain an `<actions>` element. If it is present, it must appear first in the list of elements under a `<menu>` element.

Let us look at an example that requires pre-processing actions in menu widgets. The screen widget `NestedSections` has four possible scenarios requiring four menu items.

In the menu widget `TopMenu`, add the following menu item:

```
<menu-item name="NestedSections" title="Nested Sections">
    <link target="NestedSections"/>
</menu-item>
```

In the file `LearningMenus.xml`, create a new menu widget `NestedSectionsMenu`:

```
<menu name="NestedSectionsMenu">
    <actions>
        <script location="component://learning/script/org/ofbiz/learning/
nestedSectionsMenu.bsh"/>
    </actions>
    <menu-item name="OuterTrueInnerTrue" title="Outer true -- Inner
true">
        <link target="NestedSections?outer=true&inner=true"/>
    </menu-item>
```

Other View Element Types in Screen Widgets

```
<menu-item name="OuterTrueInnerFalse" title="Outer true -- Inner  
false">  
    <link target="NestedSections?outer=true&&inner=false"/>  
</menu-item>  
<menu-item name="OuterFalseInnerTrue" title="Outer false -- Inner  
true">  
    <link target="NestedSections?outer=false&&inner=true"/>  
</menu-item>  
<menu-item name="OuterFalseInnerFalse" title="Outer false -- Inner  
false">  
    <link target="NestedSections?outer=false&&inner=false"/>  
</menu-item>  
</menu>
```

In the file `LearningScreens.xml`, create a new screen widget `NestedSectionsWithMenu` to wrap the decorator `CommonLearningDecorator` around the screen widget `NestedSections`:

```
<screen name="NestedSectionsWithMenu">  
    <section>  
        <actions><set field="tabButtonItem" value="NestedSections"/></  
actions>  
        <widgets>  
            <decorator-screen name="CommonLearningDecorator">  
                <decorator-section name="body">  
                    <include-menu name="NestedSectionsMenu"  
                        location="component://learning/widget/  
                            learning/LearningMenus.xml"/>  
                    <include-screen name="NestedSections"/>  
                </decorator-section>  
            </decorator-screen>  
        </widgets>  
    </section>  
</screen>
```

and change the view map `NestedSections` in the file `controller.xml` to this:

```
<view-map name="NestedSections" type="screen"  
    page="component://learning/widget/learning/  
        LearningScreens.xml#NestedSectionsWithMenu"/>
```

In the menu widget `NestedSectionsMenu`, we have an `<actions>` element containing a reference to a script file `nestedSectionsMenu.bsh`. In the folder `$(component:learning)\webapp\learning\WEB-INF\actions\learning`, create a new file `nestedSectionsMenu.bsh` and enter into it this:

```
String outer = request.getParameter("outer");  
String inner = request.getParameter("inner");
```

```

String fieldName = "tabButtonItem";
if ("true".equalsIgnoreCase(outer)) {
    if ("true".equalsIgnoreCase(inner))
        context.put(fieldName, "OuterTrueInnerTrue");
    else
        context.put(fieldName, "OuterTrueInnerFalse");
}
else {
    if ("true".equalsIgnoreCase(inner))
        context.put(fieldName, "OuterFalseInnerTrue");
    else
        context.put(fieldName, "OuterFalseInnerFalse");
}

```

Notice that we are using the same variable name `tabButtonItem` that was also used in the top-level menu `TopMenu`. However, this is really a different variable. The context of the parent, which is the screen widget `NestedSectionsWithMenu`, does have a similarly named variable `tabButtonItem` that was assigned a value of `NestedSections`. When that screen widget includes the menu widget `NestedSectionsMenu`, the screen widget's context is saved in a stack ("pushed into a stack" in technical speak) before creating a new context for the menu widget.

The `NestedSections` screen is finally clean and easy to experiment with. Prior to this, we've had to manually send GET parameters `outer` and `inner` to play with this screen. Now, we just need to click on the menu items:



The above example can be rewritten with a sub-menu of two menu items serving as toggles for the two request parameters `outer` and `inner`. However, this can make the menu widget code large and cumbersome. Each toggle button must be coded with four conditional `<menu-item>` elements, each serving one of the four possible scenarios. In the case above, a total of eight `<menu-item>` elements must be written. If you're feeling like a lot of typing, you can try this out. Use the menu widget `ConditionalScreenMenu` as a guiding example.

Finally, as an exercise, try to put the screen widget `CompoundedScreen` under the menu widget `TopMenu` as well. Here are a few tips. The screen widget `CompoundedScreen` needs to be changed to use the decorator `CommonLearningDecorator` instead of `xhtml-decorator` directly. The screen widget also needs an `<actions>` element. The menu widget `TopMenu` needs a new `<menu-item>` added to it.

FreeMarker

FreeMarker is a very powerful and versatile template engine written in Java, which for the purposes of the following examples produces XHTML. We can learn everything there is to know about using FreeMarker from the **FreeMarker Manual** (<http://freemarker.org/docs/index.html>). The use of FreeMarker in OFBiz is widespread and incredibly useful; it is also discussed in following chapters.

As Decorator Templates

When used in the context of a screen widget, FreeMarker files gain the ability to include screens and include sections, which are equivalent to screen widget's `<include-screen>` and `<decorator-section-include>` elements.

An example of rendering a screen widget within a FreeMarker file was encountered in Chapter 3 when we tidied up the mess we had made in the core OFBiz code in the party component. The file `${webapp:party}\party\editcontactmech.ftl` used the construct:

```
 ${screens.render("component://learning/widget/learning/
LearningScreens.xml#\editcontactmech.extend") }
```

We now look at how FreeMarker files can be used as decorator templates similar to the decorator screen widgets. Recall that we were using a single `header.ftl` file before we split it up into two files, `header1.ftl` and `header2.ftl`, in order to accommodate the variable title in the XHTML `<title>` element. This was done in the section called *Multiple Content Slots* in Chapter 3. Edit the file `header.ftl` and replace the XHTML `<title>` element with this:

```
<title>${sections.render("title")}</title>
```

and copy over from `header1.ftl` and insert above the XHTML `<title>` element this:

```
<script language="javascript" src="/images/selectall.js"
type="text/javascript"></script>
<link rel="stylesheet" href="/images/maincss.css" type="text/css"/>
```

In the file `LearningScreens.xml`, change the decorator `xhtml-decorator` to use the single FreeMarker decorator template. Replace this:

```
<include-screen name="header1"/>
<decorator-section-include name="title"/>
<include-screen name="header2"/>
```

with this:

```
<platform-specific><html>
    <html-template-decorator location="component://learning/webapp/
        learning/includes/header.ftl">
        <html-template-decorator-section name="title">
            <decorator-section-include name="title"/>
        </html-template-decorator-section>
    </html-template-decorator>
</html></platform-specific>
```

Note how similar the structure of a FreeMarker decorator is to a screen widget decorator's. The `<html-template-decorator>` element is equivalent to a `<decorator-screen>` element. The `<html-template-decorator-section>` element is equivalent to a `<decorator-section>` element. The FreeMarker code `${sections.render("title")}` is equivalent to `<decorator-section-include name="title"/>`.

Fire an OFBiz http request `CompoundedScreen` to `webapp learning`. There should be no visible difference.

In OFBiz the `<html-template-decorator>` element is only used once. Although this may appear from time to time, its use is not encouraged. To include the sections in the decorator template just use the `render(sectionName)` method `sections` object, for example: `${sections.render("main")}`. This leads to clearer and more efficient code.

Displaying Dynamically Created List Variables

One area where FreeMarker wins over static XHTML is where lists of variables need to be displayed. This is used very often in OFBiz. Here is an example. Edit `${webapp:learning}\includes\header.ftl`, and replace this:

```
<script language="javascript" src="/images/selectall.js"
    type="text/javascript"></script>
<link rel="stylesheet" href="/images/maincss.css" type="text/css"/>
```

with this:

```
<if layoutSettings.javaScripts?has_content>
    <list layoutSettings.javaScripts as javaScript>
        <script language="javascript"
            src="<@ofbizContentUrl>${javaScript}<@ofbizContentUrl>"
            type="text/javascript"></script>
    </list>
</if>
<if layoutSettings.styleSheets?has_content>
    <list layoutSettings.styleSheets as styleSheet>
```

```
<link rel="stylesheet"
      href="<@ofbizContentUrl>${styleSheet}</@ofbizContentUrl>"
      type="text/css"/>
</#list>
<#else>
<link rel="stylesheet" href="<@ofbizContentUrl>/images/maincss.css
</@ofbizContentUrl>" type="text/css"/>
</#if>
```

In the file `LearningScreens.xml`, edit the screen widget `xhtml-decorator` to add this:

```
<actions>
<set field="layoutSettings.styleSheets[] "
      value="/images/maincss.css"/>
<set field="layoutSettings.javaScripts[] "
      value="/images/selectall.js"/>
</actions>
```

Iterating Through the List

The above actions set the location as a String for one style sheet and one JavaScript file into an array, which is then iterated through by the FreeMarker template. We could have placed the location of as many files as we liked into these arrays, and the XHTML would have been created accordingly.

For example, in the above actions we could have put:

```
<set field="layoutSettings.styleSheets[] "
      value="/images/maincss1.css"/>
<set field="layoutSettings.styleSheets[] "
      value="/images/maincss2.css"/>
<set field="layoutSettings.styleSheets[] "
      value="/images/maincss3.css"/>
```

and the FreeMarker code in the template:

```
<#list layoutSettings.styleSheets as styleSheet>
<link rel="stylesheet"
      href="<@ofbizContentUrl>${styleSheet}</@ofbizContentUrl>"
      type="text/css"/>
</#list>
```

will ultimately output the following XHTML:

```
<link rel="stylesheet" href="/images/maincss1.css" type="text/css"/>
<link rel="stylesheet" href="/images/maincss2.css" type="text/css"/>
<link rel="stylesheet" href="/images/maincss2.css" type="text/css"/>
```

This handy features allows us to vary the number of variables passed into the template in different screens without us having to alter the template code.

Bringing it All Together

Now that we are coming to the end of our exploration into screen widgets and understand a bit more about how the different parts hang together, let's start to use the existing screen widgets that have been designed to give the components in OFBiz a consistent "look and feel" and page structure. These widgets, since they are common to webapps in both the applications folders (order, party) and the framework folders (webtools), can be found in the common folder `framework\common\widget`. Take a look at the `GlobalDecorator` screen widget inside the `CommonScreens.xml` file. This is the top level decorator and is responsible for the addition of the stylesheets, adding the header, adding the top tabs, and the overall structure of the page. There is the inclusion of the `messages.ftl` template, which is responsible for displaying the error and success messages. The `<container>` tags are outputted as XHTML `<div>` tags, with the `id` attribute remaining as `id` and the `style` attribute becoming `class`. So `<container style="centerarea">` becomes `<div class="centerarea">`. As such, we would expect to see a `.centerarea` style class defined in the file `framework\images\maincss.css`.

To change our learning component, we are going to take a look at how an existing component is built and apply the necessary changes to our `LearningScreens.xml` file. First of all, fire an http request to `partymgr` and login using the **admin, ofbiz** user. The main screen of the Party Manager is the **Find Party** screen. We are going to see how this page is built. Our first step is to open the Party Managers webapp's `controller.xml` file and check which screen is referenced in the `<view-map name="main">` element. This leads us to the screen: `component://party/widget/partymgr/PartyScreens.xml#findparty`.

This screen is a simple FreeMarker template that is wrapped in a Decorator called `main-decorator`, you will notice that its location is a parameter `${parameters.mainDecoratorLocation}`. This parameter is specified in the `web.xml` file for this webapp:

```
<context-param>
    <param-name>mainDecoratorLocation</param-name>
    <param-value>component://party/widget/partymgr/
        CommonScreens.xml</param-value>
    <description>The location of the main-decorator screen to use for
        this webapp; referred to as a context variable in
        screen def XML files.</description>
</context-param>
```

Let's start by adding a similar entry into our learning webapp's web.xml file:

```
<context-param>
    <param-name>mainDecoratorLocation</param-name>
    <param-value>component://learning/widget/learning/
        CommonScreens.xml</param-value>
    <description>The location of the main-decorator screen to use for
        this webapp; referred to as a context variable in
        screen def XML files.</description>
</context-param>
```

If we want to quickly change which main-decorator out of all screen widgets to use, we can change the location in just one place. The main-decorator has been separated out into a separate file because it is used by all of the other Screens.xml files within this component. The file component://learning/widget/learning/CommonScreens.xml does not yet exist, so create that.

Open the component://party/widget/partymgr/CommonScreens.xml file and copy and paste the whole main-decorator widget into our newly created CommonScreens.xml file.

There are a few new concepts here that will be explained as we work our way through this decorator.

The User-Interface Labels

We can see there are a number of property-map resources defined (PartyUiLabels, AccountingUiLabels). These refer to properties files found in the component's config directory. On opening the party\config directory we can see that there are a number of PartyUiLabels_xx.properties files and one PartyUiLabels.properties file. The files containing an underscore are files that have been translated into a different language and are used automatically, dependant on the user's locale. This is explored further in the chapter. Note that in current developed version of OFBiz all *UiLabels*.properties files have replaced by *UiLabels.xml files.

The line `<property-map resource="PartyUiLabels" map-name="uiLabelMap" global="true"/>` places the referenced properties file into a map called uiLabelMap in the global context. This makes the properties file available to all screen widgets, (including form and menu widgets), FreeMarker templates and their associated action scripts that are wrapped by this decorator.

When we want to use one of these properties, we simply pull the value from the map by using uiLabelMap.propertyName. For example, if we want to display the Company Name in a Freemaker template, we would put \${uiLabelMap.PartyCompanyName}.

Our component does not yet have a uiLabels properties file. In a new directory \${component:learning}\config, create a new file called LearningUiLabels.properties.

In it, place four properties:

```
LearningComponent=Our Learning Application  
LearningCompanyName=Learning Company Name  
LearningCompanySubtitle=Learning Company Subtitle  
LearningMain=Main
```

The name of this file is very important. For the framework to recognize it as a UiLabels.properties file, it has to follow the correct naming convention. Also, as we do not need to give a location to the files in the <property-map> element, it is important that the names are unique. We must make sure there is only one file called LearningUiLabels.properties in the whole project.

This config directory must be on the classpath for it to be recognized so open the file \${component:learning}\ofbizComponent, and add the line:

```
<classpath type="dir" location="config"/>
```

immediately underneath the other <classpath> elements.

Our learning component is much more simple than the party manager component from where we copied this screen widget from and does not need all of these properties. Delete five of the <property-map> elements, leaving the reference to CommonUiLabels and replace them with just our newly created uiLabels:

```
<property-map resource="LearningUiLabels" map-name="uiLabelMap"  
global="true"/>
```

We can now change the references to PartyCompanyName and PartyCompanySubtitle to match our new LearningCompanyName and LearningCompanySubtitle properties.

The addition of the extra stylesheets and JavaScripts are not needed and can be deleted and the activeApp can be set to learning.

The contents of Commonscreens.xml should now simply be:

```
<screens xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/  
widget-screen.xsd">  
<screen name="main-decorator">  
<section>  
<actions>
```

```
<!-- base/top/specific map first, then more common  
map added for shared labels -->  
<property-map resource="LearningUiLabels"  
map-name="uiLabelMap" global="true"/>  
<property-map resource="CommonUiLabels" map-name="uiLabelMap"  
global="true"/>  
    <set field="layoutSettings.companyName"  
        from-field="uiLabelMap.LearningCompanyName"  
        global="true"/>  
    <set field="layoutSettings.companySubtitle"  
        from-field="uiLabelMap.LearningCompanySubtitle"  
        global="true"/>  
    <set field="layoutSettings.headerImageUrl"  
        value="/images/ofbiz_logo.jpg" global="true"/>  
    <set field="activeApp" value="learning"  
        global="true"/>  
    <set field="appheaderTemplate"  
        value="component://learning/webapp/  
            learning/includes/appheader.ftl"  
        global="true"/>  
</actions>  
<widgets>  
    <include-screen name="GlobalDecorator"  
        location="component://common/widget/  
            CommonScreens.xml"/>  
</widgets>  
</section>  
</screen>  
</screens>
```

Adding the appheader

You may have noticed that the appheaderTemplate location has been changed. The GlobalDecorator needs this value to be set or it will throw an exception. In the includes directory add the appheader.ftl file and enter into it:

```
<#assign selected = headerItem?default("void")>  
<div id="app-navigation">  
    <h2>${uiLabelMap.LearningApplication}</h2>  
    <ul>  
        <li<#if selected == "main" class="selected"></#if>>  
            <a href="@ofbizUrl>main<@ofbizUrl">  
                ${uiLabelMap.CommonMain}</a></li>  
  
        <#if userLogin?has_content>  
            <li class="opposed"><a href="@ofbizUrl>logout<@ofbizUrl">  
                ${uiLabelMap.CommonLogout}</a></li>  
        <#else>  
            <li class="opposed"><a href="@ofbizUrl>${checkLoginUrl?>
```

```

        if_exists}><@ofbizUrl>">${uiLabelMap.CommonLogin}</a></li>
    </#if>
</ul>
<br class="clear"/>
</div>

```

We must now tell our existing screen widgets in the `LearningScreens.xml` file to use this new `main-decorator`.

Simply find all instances of:

```
<decorator-screen name="CommonLearningDecorator">
```

and replace them with:

```
<decorator-screen name="main-decorator"
    location="${parameters.mainDecoratorLocation}">
```

As we have added and changed some `.properties` files and added to the `web.xml` file, we must restart OFBiz to see these changes take effect. Once restarted, fire an http request to the webapp `partymgr` and take a look at the applications tabs along the top of the screen.



Clicking on the **Learning** tab will still give a white screen and an error in the logs. The OFBiz framework automatically produces the tabs for each component, and it automatically creates the links pointing to `main`. We must therefore define a request-map called `main` in our `controller.xml` file by adding:

```
<request-map uri="main">
    <response name="success" type="view" value="main"/>
</request-map>
```

And add the `view-map`:

```
<view-map name="main" type="screen"
    page="component://learning/widget/learning/
    LearningScreens.xml#SimplestScreenWithMenu"/>
```

Since the `main` request is usually the first port of call when accessing the webapps, it is common practice for it to be the first request-map element in `controller.xml` files. Once this is in place, select the **Learning** tab.

As soon as you enter the **Learning Application** the tabs disappear. The tabs are only displayed to users who are logged in, and this means logged into the individual webapp. There is no way to log into the application as a whole, only each component individually.

In the next chapter we will be exploring how we can add login functionality to our Learning component.

Summary

With that, we've covered the most developed and most stable parts of OFBiz widgets and we now have a working OFBiz application. The widgets we have covered, screen, form, and menu, are what we will be using almost all of the time. For more creative and complex user-interfaces, we fall back on FreeMarker.

As powerful as FreeMarker already is, the OFBiz Widget Engine integrates with it such that it gains the invaluable organizational structure of OFBiz widgets.

FreeMarker under OFBiz widgets can utilize the Widget Engine's decorator model giving the ability to neatly compose larger screens with smaller screens or sections, and even some access to the database (covered later in the book).

In this chapter, we looked at:

- Menu widgets, and creating them via `<menu>` elements within a `<menus>` element, like how we placed `<form>` elements within a `<form>` element.
- Creating our first menu widget in the file `LearningMenus.xml`.
- Embedding or including menu widgets into screen widgets via `<include-menu>` elements.
- Creating sub-menus with menu widgets, simply by placing sub-menus under top-level menus.
- Pre-processing for menu widgets, much like the `<actions>` element for screen widgets. The example given serves to streamline conditionals in menu widgets with succinct pre-processing scripts.
- OFBiz's widget (View component) integration with FreeMarker via elements `<platform-specific>` and `<html>`.
- Using FreeMarker files as decorator templates via elements `<html-template-decorator>` and `<html-template-decorator-section>`.
- Displaying dynamically created list variables in FreeMarker using the `<#list ... as ...>` FreeMarker construct.
- Reusing existing screen widgets and creating an OFBiz application from existing screens.

That wraps up our study of the View component of OFBiz. In the next chapter, we will look at the flow through OFBiz – The Controller.

6

The Controller

Every webapp has a controller defined in a file `web.xml`. The controller handles all incoming requests from the end-user, and that's how a webapp "listens to" and "responds to" the end-user. The controller in an OFBiz webapp is called a **control servlet**.

In this chapter, we will be looking at:

- Understanding, defining, and using a control servlet
- Understanding and defining the utility objects that OFBiz automatically attaches to every end-user request
- Creating a complete flow from receiving an end-user request to determining a response
- Working with various types of security and responses

How OFBiz Hears Our Requests—The Control Servlet

For each webapp, OFBiz employs a **Front Controller** pattern where a single component serves as a centralized access point and controls all access to the webapp. This allows OFBiz to use a uniform processing workflow for all requests fired to any webapp in the entire application, thereby resulting in the consistent reuse of a lot of code and logic in OFBiz.

In this section, we will deal with the setting up of a control servlet.

A typical processing workflow for a request could be like this:

- Check that the request is valid
- Perform security checks

- Log request (or visit)
- Perform any preprocessing specified
- Look up the defined processing for the request (defined by a `request_map` in `controller.xml`)
- Process the specified event (if any) for the request
- Determine the response (depending on prior event, if there was one)
- Render the response to the web browser

As we can see there is quite a lot of standard processing for every request. It makes good sense and is cost efficient to use the Front Controller pattern. The single mechanism (per webapp) employed by OFBiz to serve in the Front Controller pattern is called a control servlet. The control servlet is programmed to always perform the above workflow without further instructions from the incoming requests. It's like a smart front-desk receptionist being able to say "Good Afternoon Sir" or "Good Morning Madam" without needing incoming requesters to indicate how they would like to be addressed. Using a control servlet like this greatly simplifies the format of incoming requests.

There's no need to understand the Front Controller pattern for this book, except to know that one and only one control servlet is needed for each web application in OFBiz. For those who would like to know more about control servlets, more information can be found on the Front Controller Pattern on the Java web site at <http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html>.

Defining a Control Servlet for a Webapp

The control servlet is defined in a webapp's `web.xml` by a `<servlet>` element. Let's take a look at an example in the file `${webapp:learning}\WEB-INF\web.xml`:

```
<servlet>
  <servlet-name>ControlServlet</servlet-name>
  <servlet-class>org.ofbiz.webapp.control.ControlServlet
  </servlet-class>
</servlet>
```

The value of the `<servlet-name>` sub-element acts as a handle for referring to the defined servlet. It must be unique within the file `web.xml`. If there are more than one control servlets defined, then only the first will be recognized.

The `<servlet-class>` sub-element defines the implementation of the control servlet. In this case, we want to use OFBiz's implementation that is in the class `org.ofbiz.webapp.control.ControlServlet`.

Using the Control Servlet

By default, all request string patterns are handled by the embedded Tomcat server. We need to define a request string pattern that the control servlet must intercept and process. In other words, we need to map a URL pattern to a control servlet using a `<servlet-mapping>` element. In the file `${webapp:learning}\WEB-INF\web.xml`, we see an example of this:

```
<servlet-mapping>
    <servlet-name>ControlServlet</servlet-name>
    <url-pattern>/control/*</url-pattern>
</servlet-mapping>
```

We can easily get the control servlet to intercept request string patterns of `/SomeOtherName/*`, so that a typical OFBiz https request `SomeRequest` to `webapp learning` will have a URL of `https://localhost:8443/learning/SomeOtherName/SomeRequest`. However, the convention in OFBiz is to use the pattern `/control/*`.

Funnelling All Requests to the Single Control Servlet

Since the concept of the Front Controller pattern is to have a single point of entry through a single control servlet, OFBiz has a filter entity that can restrict the embedded Tomcat server to accept only URL patterns starting with `control/`. That is, URLs that call on the single control servlet in a webapp. We define a filter with a `<filter>` element, like in the file `${webapp:learning}\WEB-INF\web.xml`:

```
<filter>
    <filter-name>ContextFilter</filter-name>
    <filter-class>org.ofbiz.webapp.control.ContextFilter</filter-class>
    <init-param>
        <param-name>allowedPaths</param-name>
        <param-value>/control:/index.html</param-value>
    </init-param>
</filter>
```

In the above example, we defined a filter with an implementation that is OFBiz's class `org.ofbiz.webapp.control.ContextFilter`. Like using a `<servlet>` with `<servlet-mapping>`, we need to use the above filter with:

```
<filter-mapping>
    <filter-name>ContextFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

That means pump every end-user request through that filter.

The filter implementation that is `ContextFilter` is designed to allow only URLs that fit the pattern in the value for the parameter `allowedPaths`. Multiple patterns can be specified, separated by a ":" character. The filter above is configured to allow request strings starting with `control`, a specific request string `index.html`, and nothing else.

There is one quirk with the `allowedPaths` parameter of `ContextFilter`. It only considers the first level of a request path, where the depth of a request path like `/level1/level2/SomeFile.html` is 3 while `/SomeFile.html` is 1. Therefore, it is not possible to allow `/level1/level2/index.html` without also allowing `/level1/AnyOtherFiles.html`. Specifying an allowed path like `/level1/level2`, or any path with depth greater than one, is meaningless to the `ContextFilter`. It also follows that to allow a request `/level1/level2/index.html`, we need to specify an allowed path of `/level1`; nothing else works.

There are only two types of allowed paths that `ContextFilter` can understand. One is the single-depth path of the form `/AnyPathName`, which means to allow for requests that start with `/AnyPathName`. The other is the filename of a specific file at the root path, for example `/AnyFile.html`. Specifying an allowed path of `"/"` (root, depth of 0) does not allow access to all files under `"/"`; it does nothing, in fact.

For the purposes of `ContextFilter` in OFBiz, which is to funnel all requests through the single control servlet in a webapp, `ContextFilter`'s current behavior with `allowedPaths` parameter is adequate.

Defining Needed Utility Objects for the Control Servlet

There are two utility objects we need to define for a control servlet. There are altogether three necessary utility objects for request handling in general, but only two can and need to be defined in the `web.xml` file.

GenericDelegator Object

`GenericDelegator` objects are used for database access. The ways in which the framework makes this object available for us to use in our code will be fully explained later.

One of the activities of the Control Servlet is to log requests (or visits) from the end-user. The control servlet records each request in the database. Therefore, the control servlet needs access to the database.

To define a `GenericDelegator` object for a Control Servlet, we define a parameter named `entityDelegatorName` like this:

```
<context-param>
    <param-name>entityDelegatorName</param-name>
    <param-value>default</param-value>
</context-param>
```

Another name for "database-related objects" is "entity", hence the parameter name `entityDelegatorName`.

OFBiz has a single pre-defined delegator named `default`. We don't need to know what a delegator is for now, except that it is like a worker to whom we delegate all database-related tasks. The delegator then delegates database-related tasks to the correct datasource helper.

Note that a `ContextFilter` object will always load up an entity delegator even if we hadn't specified one via the `entityDelegatorName` parameter. By default, a `ContextFilter` object will load up an entity delegator by the name of `default`. Database access is so fundamentally needed by the Control Servlet (and by OFBiz) that a `ContextFilter` object will not do without one. Try removing the parameter `entityDelegatorName` in the file `${webapp:learning}\WEB-INF\web.xml`, restart OFBiz, and see that everything still works like before.

The GenericDispatcher Object

`GenericDispatcher` objects are related to OFBiz's Service Engine, and will be covered in more detail later in the book. For now, we just need to know that the `ContextFilter` object requires an accompanying `GenericDispatcher` object to be defined.

This coupling between `ContextFilter` and `GenericDispatcher` is too tight and makes it impossible for OFBiz to be deployed without the Service Engine. The examples in this book so far have not required the Service Engine to be involved. However, the Service Engine is an important and powerful aspect of OFBiz, and there are few, if any, real-world deployment scenarios where the Service Engine should be omitted.

To define a `GenericDispatcher` object for a control servlet, we define a parameter named `localDispatcherName` like this:

```
<context-param>
    <param-name>localDispatcherName</param-name>
    <param-value>learning</param-value>
</context-param>
```

For now, we just need to know that the value of parameter `localDispatcherName` should match the mount point. So, for webapp `partymgr`, the value of `localDispatcherName` for the control servlet of that webapp will be `partymgr`.

Some Background on Servlets

We see lots of instances of the string `servlet` in the file `web.xml`. OFBiz is run on an embedded servlet "container", where "container" is simply a containing or encapsulating entity that manages the servlets placed in it. A servlet is "an object that receives a request and generates a response based on that request". That definition also applies to a server. So what's the difference between a server and a servlet?

A servlet is a fine-grained server, a smaller and more agile entity. Historically, a typical server application was a single "listen-process-respond" loop that, especially if caused to fail by a bad incoming request, would often crash completely without the possibility for an automatic recovery. Today, the "server" has been replaced by a "container" that manages one or more "servlets". If a contained servlet fails, a new one can be created to take its place.

Programming a Control Servlet

In this section, we will deal with programming or instructing the control servlet. A control servlet reads the file `controller.xml`, which we have already been using a great deal to add `request-maps` and `view-maps` and publish our screens.

These files may be referred to as "site config" files. The root element of the controller files is the `<site-config>` element and they are indeed the files responsible for configuring the flow through the site and contain instructions for the control servlet they are related to.

Let's continue instructing the control servlet by adding login functionality.

Logging into Our Learning Application

Fire an http request `main` to the webapp `partymgr`, and once again click the **Learning** tab.

As we have already seen in previous examples, as soon as we enter the **Learning** Application the tabs disappear. The tabs are only displayed to users who are logged in, and this means logged into the individual webapp. There is no way to log into the application as a whole, only each webapp individually. However, if you were to go back to the party manager, then click on the **Facility** tab, you are not sent back to the login screen and asked for your login details again. So what's the difference between clicking on our **Learning** tab and clicking on the **Facility** tab?

Clicking on the **Facility** tab will send us to the URL `https://localhost:8443/facility/control/main?externalLoginKey=EL*****` (where the asterisks represent a random number). Notice that the framework has added a parameter on the query string called `externalLoginKey`. This parameter is the key that the framework uses to lookup to see if this current session is logged in or not.

You may notice that if you clicked on the **Learning** tab the `externalLoginKey` parameter had already been added. All that we need to do is to tell our webapp that we want to check for this `externalLoginKey` parameter and that this should be the first thing we do when we start to process the request.

To do this we add a new element into the controller, immediately underneath the `<handler>` elements and above the `<request-maps>`:

```
<preprocessor>
    <event type="java"
        path="org.ofbiz.webapp.control.LoginWorker"
        invoke="checkExternalLoginKey"/>
</preprocessor>
```

The `<preprocessor>` element specifies all events that are to be run on every request prior to security.

There is also a `<postprocessor>` element which can be added. This specifies all events that are to be run on every request after security.

With this in our site config, click on the **Learning** tab again from any of the other webapps and notice that the tabs are now visible. We have been automatically logged into the **Learning Application**.

We may want to login directly to the Learning application without having to go to the Party Manager and then clicking on the tab. We must now tell the webapp the location of the login screens by adding the security mappings to the `controller.xml`. Immediately below the `main` request-map add:

```
<!-- Security Mappings -->
<request-map uri="checkLogin" edit="false">
    <description>Verify a user is logged in.</description>
    <security https="true" auth="false"/>
    <event type="java" path="org.ofbiz.webapp.control.LoginWorker"
        invoke="checkLogin" />
    <response name="success" type="view" value="main" />
    <response name="error" type="view" value="login" />
</request-map>

<request-map uri="login">
    <security https="true" auth="false"/>
```

```
<event type="java" path="org.ofbiz.webapp.control.LoginWorker"
       invoke="login"/>
<response name="success" type="view" value="main"/>
<response name="error" type="view" value="login"/>
</request-map>

<request-map uri="logout">
    <security https="true" auth="true"/>
    <event type="java" path="org.ofbiz.webapp.control.LoginWorker"
           invoke="logout"/>
    <response name="success" type="request" value="checkLogin"/>
    <response name="error" type="view" value="main"/>
</request-map>
<!-- End of Security Mappings -->
```

and add the view-map:

```
<view-map name="login" type="screen" page="component://learning/
widget/learning/CommonScreens.xml#login"/>
```

The last step is to create the simple login screen in `CommonScreens.xml`:

```
<screen name="login">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <platform-specific>
                        <html><html-template
                            location="component://common/
                            webcommon/login.ftl"/></html>
                    </platform-specific>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

We can now click the logout and login to the **Learning Application**. To test this, click the **Logout** button on the right-hand side of the screen and then immediately log back in again by entering the `admin, ofbiz` user name and password. We will be covering much more on logging in and all aspects of OFBiz security in a later chapter.

Specifying Handlers

The ability to specify "handlers" in site config files make for a pluggable architecture. That is the ability to plug-in our own custom handlers if need be.

We can specify handlers for two types of programming instructions in site config files: `request` and `view`.

A `request` instruction defines a particular request handling procedure that we want our control servlet to execute upon receiving the request. It is defined by a `<request-map>` element. A `view` instruction defines a procedure of constructing and displaying a view to the end-user, and is defined by a `<view-map>` element.

The `request-map` and `view-map` instructions will be discussed in greater detail shortly. For now, we will focus on specifying handlers, because handlers need to be specified before any `request maps` or `view maps` in a site config file.

A typical handler definition for a `request map` is like this:

```
<handler name="java" type="request"
        class="org.ofbiz.webapp.event.JavaEventHandler"/>
```

And a typical `request map` using that handler is like this:

```
<request-map uri="processFirstForm">
  <event type="java" path="org.ofbiz.learning.learning.LearningEvents"
         invoke="processFirstForm"/>
  <response name="success" type="view" value="OneFormScreen"/>
</request-map>
```

Note that the sub-element `<event>` has attribute `type` with a value of `java`.

A typical handler definition for a `view map` is specified like this:

```
<handler name="screen" type="view"
        class="org.ofbiz.widget.screen.ScreenWidgetViewHandler"/>
```

The `name` attribute serves as an identifier for the handler in the site config file. For example, in the file `${webapp:learning} \WEB-INF\controller.xml`, we see the above handler named `screen`, and then a `view map` with a `type` of `screen` like this:

```
<view-map name="SimplestScreen" type="screen"
          page="component://learning/widget/learning/
                LearningScreens.xml#SimplestScreenWithMenu"/>
```

That means we are telling the control servlet to process that particular `view map` named `SimplestScreen` with a handler named `screen`.

Request Maps

Each request map, defined by a `<request-map>` element, defines a procedure with which to process an incoming request.

Firing a request to OFBiz is like knocking on a door. OFBiz first screens the request through some security procedures, some of which can be programmed or customized through the request's corresponding request map. If the request passes through security, OFBiz next performs some logic procedures to determine a response to the request. Finally, OFBiz returns the response to the end-user, often in the form of a screen widget.

Let's run through that flow in detail.

Knocking on the Right Doors

The first thing to note about the `<request-map>` element is the `uri` attribute. It tells the control servlet the exact request URI (Universal Resource Identifier, or "request name" in this case) to handle.

A `uri` attribute of `processFirstForm` will mean the request map handles OFBiz requests like `processFirstForm` or `processFirstForm?whateverParams=whateverValue`. Just as we saw in Chapter 3, the above OFBiz request is a request of the form `control/processFirstForm`.

Following this, we discuss the possible sub-elements of the `<request-map>` element. They are covered in the order they must appear, if they are to appear at all. Some sub-elements are optional.

Security—Before Answering the Door

Security related attributes can be specified for a request map via a sub-element `<security>`. This sub-element `<security>` is optional. The `<security>` sub-element can have a number of attributes. The following attributes are the ones currently used in OFBiz.

- `https`
- `auth`
- `direct-request`

The `https` Attribute

The `https` attribute, if given a value of `true`, will cause the control servlet to convert an incoming http request into an https one. Insert into `${webapp:learning}\WEB-INF\controller.xml` this request map:

```
<request-map uri="forceToHttps">
    <security https="true"/>
    <response name="success" type="view" value="forceToHttps"/>
</request-map>
```

and this view map:

```
<view-map name="forceToHttps" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#forceToHttps"/>
```

and into \${component:learning}\widget\learning\LearningScreens.xml this screen widget:

```
<screen name="forceToHttps">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <label text="This should only be viewable with an https
                        request" /><decorator-section>
                </decorator-screen>
            </widgets>
        </section>
    </screen>
```

Fire an OFBiz http request forceToHttps to webapp learning. Note how the request is converted (redirected, rather) to an OFBiz https request instead (the address bar shows an address starting with https://localhost:8443 rather than http://localhost:8080).



Omitting this attribute will mean a default value of `false`.

The auth Attribute

The `auth` attribute, if given a value of `true`, will cause the control servlet to check if the end-user is logged in. This checking is done by the control servlet firing an OFBiz https request `checkLogin` to the webapp. The request string `checkLogin` is hard coded into the control servlet. If the end-user isn't logged in, the end-user will be redirected to the login screen. Upon successfully logging in, the originally intended request will be fired. Let's look at an example.

In the file `${webapp:learning}\WEB-INF\controller.xml`, insert a new request map:

```
<request-map uri="withAuth">
    <security auth="true"/>
    <response name="success" type="view" value="withAuth"/>
</request-map>
```

and a new view map:

```
<view-map name="withAuth" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#withAuth"/>
```

In the file `LearningScreens.xml` in `${component:learning}\widget\learning`, add a new screen widget:

```
<screen name="withAuth">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <label text="This should only be viewable when logged
                        in."/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

Fire an OFBiz http request withAuth to webapp learning. Note how the login screen comes up. Log in with username **admin** and password **ofbiz**, and note how we are brought to the originally intended request withAuth.



Firing an OFBiz http request withAuth again to webapp learning will no longer bring up the login screen, since we are now logged in. Click **Logout** and note how the login screen again intercepts our OFBiz http request withAuth.

Omitting this attribute will mean a default value of `false`.

It is more common than not to use the `https` and `auth` attributes together, for instance:

```
<security https="true" auth="true"/>
```

This statement would mean the page would only be visible with a secure https request (or redirected if an http request was passed) and the page would only be visible if the user was logged in (and of course, the user would be passed to the login page if not).

The direct-request Attribute

This `direct-request` attribute, if given a value of `false`, will cause the control servlet to refuse a direct request aimed at the request map, while still allowing the request map to be called on by an internal request.

In the file \${webapp:learning}\WEB-INF\controller.xml, insert a new request map:

```
<request-map uri="noDirectRequest">
    <security direct-request="false"/>
    <response name="success" type="view" value="noDirectRequest"/>
</request-map>
```

and a view map:

```
<view-map name="noDirectRequest" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#noDirectRequest"/>
```

In the folder \${component:learning}\widget\learning, insert into the file LearningScreens.xml a new screen widget:

```
<screen name="noDirectRequest">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <label text="This should not be directly viewable with
request "noDirectRequest"."/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

Firing an OFBiz http request noDirectRequest to webapp learning will bring up an **Unknown Request Error**.

Now, we insert into the file controller.xml a new request map:

```
<request-map uri="directToReq">
    <response name="success" type="request" value="noDirectRequest"/>
</request-map>
```

Firing an OFBiz http request directToReq will now bring up the screen noDirectRequest.

Event—Determining a Response

The event sub-element points to logic and processing that determines a response to the incoming request. The sub-element can have three attributes: type, path, and invoke. The Attribute type is always needed, as it states the event handler to use. The Attributes path and invoke have different requirements, depending on the event handler used.

An invoked event must return a `java.lang.String` object. This string will determine which of the defined responses to return. Each request map can have one or more defined responses. Defining responses will be covered in the next section.

There are a number of event handlers in OFBiz. We will only consider BeanShell and Java event handlers for now, since those were the only two programming languages we have encountered so far in the book.

Java Events

When invoking Java events, we use the event handler `org.ofbiz.webapp.event.JavaEventHandler`. In our `learning` webapp's `controller.xml` file, we saw this event handler being defined:

```
<handler name="java" type="request"
         class="org.ofbiz.webapp.event.JavaEventHandler"/>
```

To invoke a Java event, the path attribute must contain the full name of the Java class that contains the Java method we want to invoke. For example `org.ofbiz.webapp.control.LoginWorker`. The invoke attribute must contain the name of the Java method to invoke. A typical example is like this:

```
<event type="java" path="org.ofbiz.webapp.control.LoginWorker"
       invoke="checkLogin" />
```

The Java method thus invoked must comply with a convention, an exact signature comprising:

- a `static` keyword that makes it a class method, not an instance method
- a return of type `java.lang.String`
- two parameters of type `javax.servlet.http.HttpServletRequest` and `javax.servlet.http.HttpServletResponse`

A typical example is in the folder `${component:webapp}\src\org\ofbiz\webapp\control` file `LoginWorker.java` at line 203:

```
public static String checkLogin(HttpServletRequest request,
                                HttpServletResponse response) {
```

To refer to Java classes without their package names, the above code snippet has accompanying import statements in lines 28-29:

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

Response—Defining Various Responses

A response is defined by a sub-element <response>. There are six types of responses possible, of which the two most used are view and request. We have so far only seen view and one instance of request in brief in this chapter.

Each <response> element has three attributes: name, type, and value.

How a Response is Chosen

The name attribute serves as an identifier for the response within a <request-map> element. The value of that attribute must be unique among all <response> elements within a request map.

Let's see how the <event> element chooses the response to return. In the folder \${webapp:learning}\WEB-INF, insert into controller.xml a new request map:

```
<request-map uri="chooseResponse">
    <event type="java"
        path="org.ofbiz.learning.learning.LearningEvents"
        invoke="chooseResponse"/>
    <response name="success" type="view" value="goodScreen"/>
    <response name="error" type="view" value="badScreen"/>
</request-map>
```

and two new view maps:

```
<view-map name="goodScreen" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#goodScreen"/>
<view-map name="badScreen" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#badScreen"/>
```

In the folder \${component:learning}\widget\learning, insert into LearningScreens.xml two new screen widgets:

```
<screen name="goodScreen">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
```

```

<decorator-section name="body">
    <label text="This is the good screen."/>
</decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>
<screen name="badScreen">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <label text="This is the bad screen."/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>

```

In the Java class `org.ofbiz.learning.learning.LearningEvents` create a new static method `chooseResponse` that is invoked in the above request-map.

```

public static String chooseResponse(HttpServletRequest request,
                                    HttpServletResponse response) {
    String responseName=(String)request.getParameter("responseName");

    if("goodResponse".equals(responseName)) {
        return "success";
    }else{
        return "error";
    }
}

```

Note how the method returns a String value. This String value must correspond with the name of an existing response `success` or `error` that is defined in the request-map.

Stop OFBiz, recompile our learning component, restart, then fire an OFBiz http request `chooseResponse?responseName=goodResponse` to see the `goodResponse` response chosen by the `<event>` element.

Fire an OFBiz http request `chooseResponse?responseName=badResponse` (or any other value for `responseName` other than `goodResponse`) to see the `badResponse` response.

The request-map can have any number of responses. Although it is always a good idea to specify the response named `error`, since the framework will return a response `error` by default if any unhandled exception occurs within the event. A good example of a request-map containing multiple responses can be found in the `ordermgr`'s `controller.xml` file. In particular the request-map named `finalizeOrder`:

```
<request-map uri="finalizeOrder">
    <security https="true" auth="true"/>
    <event type="java"
        path="org.ofbiz.order.shoppingcart.CheckOutEvents"
        invoke="finalizeOrderEntry"/>
    <response name="addparty" type="view"
        value="setAdditionalParty"/>
    <response name="customer" type="view" value="custsetting"/>
    <response name="shipping" type="view" value="shipsetting"/>
    <response name="shippingAddress" type="view"
        value="EditShipAddress"/>
    <response name="options" type="view" value="optionsetting"/>
    <response name="payment" type="request"
        value="calcShippingBeforePayment"/>
    <response name="paymentError" type="request"
        value="calcShippingBeforePayment"/>
    <response name="term" type="view" value="orderTerm"/>
    <response name="shipGroups" type="view"
        value="SetItemShipGroups"/>
    <response name="sales" type="request" value="calcShipping"/>
    <response name="po" type="view" value="confirm"/>
    <response name="error" type="request" value="orderentry"/>
</request-map>
```

Each response is determined by the outcome of the `finalizeOrderEntryMethod`. Open the class and see if you can follow through the code and find out which response is returned at which point.

Default Responses and Non-Responses

A response named `success` is a very special response in OFBiz. In the absence of an `<event>` sub-element the control servlet will automatically choose the `success` response. That is why the following works:

```
<request-map uri="main">
    <response name="success" type="view" value="main"/>
</request-map>
```

However, if an `<event>` sub-element is present and the event does not return any string value, the response will be of type `none`, which means no response at all. It'll be like having a response like this:

```
<response name="whatever" type="none"/>
```

For example, firing an OFBiz request `chooseResponse` without any parameters to `webapp learning` will trigger a non-response. No error is shown, because no error occurred. The response is just a blank screen.

We next explore four of the six types of responses in detail. Type `none` response is already described above. Type `url` is not often used. The response types are in two main categories: **views** and **requests**.

View responses usually display an output (even file objects for download) or screen to the end-user. Request responses are like "answering a question (from the end-user) with a question (to webapp or self)", or rather "responding to a request with a request (to self)".

Type "view" Responses

Type `view` responses point to view maps. The `type` attribute must have value of `view`, and the `value` attribute must be the name of an existent view map. An example is this:

```
<response name="goodResponse" type="view" value="goodScreen"/>
```

Type "request" Responses

Type `request` responses point to request maps. Such responses pass on the current request to another request map. One example we just encountered is this:

```
<request-map uri="directToReq">
    <response name="success" type="request" value="noDirectRequest"/>
</request-map>
```

which directs an OFBiz request `directToReq` to this request map:

```
<request-map uri="noDirectRequest">
    <security direct-request="false"/>
    <response name="success" type="view" value="noDirectRequest"/>
</request-map>
```

No new request is really fired. Note how the address bar still shows a URL that is an OFBiz request `directToReq`, even though the actual request map processed is at `noDirectRequest`.

Such a relay is sometimes called a **chain**. Request chains can be a useful mechanism to chain together reusable request maps and related processing.

Type "request-redirect" Responses

Unlike the type `request response`, type `request-redirect` responses actually fire off a new request. The URL in the address bar will be changed. In the folder `${webapp:learning}\WEB-INF`, insert into `controller.xml` two new request maps:

```
<request-map uri="reqRedirect">
    <response name="success" type="request-redirect" value="reqRedirectTarget"/>
</request-map>
<request-map uri="reqRedirectTarget">
    <response name="success" type="view" value="reqRedirectTarget"/>
</request-map>
```

and a new view map:

```
<view-map name="reqRedirectTarget" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#reqRedirectTarget"/>
```

In the folder `${component:learning}\widget\learning`, insert into the file `LearningScreens.xml` a new screen widget:

```
<screen name="reqRedirectTarget">
    <section>
        <actions>
            <set field="paramStr" value="${bsh:
                import org.ofbiz.base.util.*;
                UtilMisc.printMap(UtilHttp.getParameterMap(request)).
                replaceAll("&quot;\\n&quot;, &quot;&lt;br/&gt;&quot;); }"/>
        </actions>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <container><label text="Request redirected."/></container>
                    <container><label text="Params:"/></container>
                    <container><label text="${paramStr}" /></container>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

Fire an OFBiz request `reqRedirect` to `webapp learning` and note how the URL is changed to `reqRedirectTarget`, as if the end-user had fired the request `reqRedirectTarget` instead.



Request redirected.
Params:

The extra code in the screen widget `reqRedirectTarget` has to do with parameters passed in with the request, and is discussed next.

Type "request-redirect-noparam" Responses

This behaves exactly like type `request-redirect` responses, except that all parameters are dropped when the new request is fired.

First, fire to `webapp learning` an OFBiz request `reqRedirect` with parameters `someParam` and `someParam2` with values `someValue` and `someValue2`, and note how the redirected request retains the two parameters passed in.

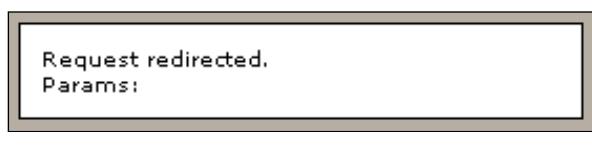


Request redirected.
Params:
`someParam --> someValue`
`someParam2 --> someValue2`

In the folder `${webapp:learning}\WEB-INF`, insert into `controller.xml` a new request map:

```
<request-map uri="reqRedirectNoparam">
    <response name="success" type="request-redirect-noparam"
              value="reqRedirectTarget"/>
</request-map>
```

Now fire to `webapp learning` an OFBiz request `reqRedirectNoparam` with the same parameters and values, and note how the redirected request has dropped all the parameters passed in.



Request redirected.
Params:

View Maps

Each view map, defined by a `<view-map>` element, defines a view that is presented to the end-user. Such a view is presented as a response to an incoming end-user request.

By now, view maps should be very familiar to us. We have used them very often since Chapter 2. There are only two types of view maps that are commonly used in OFBiz: the screen widget and the Screen FOP.

The Screen FOP will be covered later in the book. It involves generating PDF output.

The screen widget view is what we have been using. OFBiz uses the view handler `org.ofbiz.widget.screen.ScreenWidgetViewHandler` to handle such views. We typically see it being included in a site config like this:

```
<handler name="screen" type="view"
         class="org.ofbiz.widget.screen.ScreenWidgetViewHandler"/>
```

A typical screen widget view is like:

```
<view-map name="main" type="screen"
          page="component://learning/widget/learning/
                LearningScreens.xml#main"/>
```

The `name` attribute is a unique handle with which we refer to view maps (such as when we refer to them via request map responses). The value must be unique such that there are no view maps with the same name in a single webapp.

The `type` attribute specifies which view handler to use. In the case above, having a value of `screen` will use the screen widget view handler.

The `page` attribute states the location of the screen widget to load.

For the purposes of screen widgets, the attribute `content-type` is never used. Or rather, it is always forced to `text/html`. For Screen FOPs, which will be covered later, the `content-type` is `application/pdf`, which causes the internet browser to invoke an embedded PDF viewer to display the returned PDF content.

Summary

In this chapter, we had looked at:

- The control servlet and its roles and services.
- Defining a control servlet with a `<servlet>` element in a `web.xml` file.
- Using a control servlet with a `<servlet-mapping>` element.
- Confining end-user requests to the control servlet with a `<filter>` element.
- Programming the filter with the parameter `allowedPaths`.
- Defining utility objects `GenericDelegator` and `GenericDispatcher` for the control servlet.
- Programming the control servlet with a site config file `controller.xml`.
- Creating conventional view maps "main" and "login".
- Specifying event and view handlers with the `<handler>` element.
- Defining request processing flow with `<request-map>` elements.
- Defining security for request maps with a `<security>` element.
- Determining a response to an incoming end-user request with a `<event>` element.
- Defining various types of responses with `<response>` elements.
- Defining views with `<view-map>` elements.

That about covers everything we need to know to use the site config and to control the flow through our application, from defining it in the `web.xml` file to programming it in the `controller.xml` file.

In the next chapter, we will be looking at the Model that drives OFBiz and begin to understand the concepts of Entities and View Entities.

7

Entities, View Entities, and Extended Entities

We now come to the **Model** component of OFBiz—the "M" of the MVC. The Model component of the MVC is also known as the database. While the View portion takes care of presenting forms and user-interfaces and feedback to the end-user, the database stores all the information that the application is designed to store. If our application helps us as a business to serve our customers better, then our database will contain possibly a lifetime of data about our customers.

The Entity Engine in OFBiz is declarative. That is, it does not involve speaking in the native languages of many types of database systems that OFBiz supports. This means you only need to learn one Entity Engine in OFBiz, and you will be able to leverage any of the supported database systems.

In this chapter, we will be looking at:

- Required administrative Entity Engine concepts (datasources and delegators)
- The anatomy of an Entity
- Creating and using our first Entity
- Relations between Entities, and the corresponding Foreign Keys
- Indices on Entity Fields
- Relation types— "one" and "many" and "one-nofk"
- The anatomy of a View Entity
- Linking Entities (or even other View Entities) in a View Entity
- Two types of Linking: Inner Joins and Outer Joins
- Functions (example arithmetic, counting) on View Entity fields
- Grouping for summary views

- Complex aliases (where functions don't cut it)
- Extending Entities

Entities

Entities in OFBiz, or rather in databases, are the basic units of a Model in the MVC framework. Simply speaking, an entity is a single database table. A table contains information about an entity in the real world, such as a person. Therefore, a table named Person would contain fields that describe a person.

The Structure of the Data Model

The OFBiz data model is far too big to be explained in this book. The design and concepts of most parts of the data model are based on models documented by Len Silverstone in *The Data Model Resource Book, Volumes 1 and 2*. This book clearly explains the underlying data structure and by reading it you will gain a better knowledge of the model that drives OFBiz.

Referencing Fields of an Entity

We generally refer to a field of an entity like this: <EntityName>. <FieldName>. The field planet in the entity PostalAddress will be referred to as PostalAddress.planet.

OFBiz Uses Relational Database Management Systems

OFBiz uses RDBMSs. RDBMSs use the "relational model". The relational model uses tables that may be related to one another. A simple example is a Person table that contains records having the field postalAddress. That field could point to a particular record in another table PostalAddress. In that way, we say that a person can be related to (or "can have") a postal address. Note how the table Person contains only fields that describe a person, and not fields like street that describe a postal address. That is how an entity should be defined, to describe only the entity itself (say Person) and not possibly associated entities (say PostalAddress).

Curious Trivia about the Relational Model

The word "relational" in a relational model does not mean, as you would probably expect it to, that tables relate to one another. Instead "relational" is a mathematical term for a "table". A database management system based on the relational model deals with tables. It just so happens that these tables can be related to one another to form complex structures (like a person having one or more postal addresses, each one having several pieces of furniture). The word "relationship" in an **Entity-Relationship Model (ERM)** refers to the relationship between entities. An ERM is a representation of structured data.

Entity Engine Concepts

There are a few concepts we need to quickly learn about before we proceed, concepts related to databases in OFBiz.

Datasources

A datasource is simply a "source of data". In OFBiz Derby is used as the default RDBMS. There can be multiple database schemas in one database instance, each schema housing a separate database of entities (say schemas `people_records` and `address_records` house records for people and addresses, respectively). For database systems that do not support schemas, separate database instances will need to be created instead for that same purpose.

Datasources are defined and configured in `${component:entity}\config\entityengine.xml`. The component entity is in the folder `${OFBizInstallFolder}\framework\entity`. The default datasource used in OFBiz is `localderby`:

```
<datasource name="localderby"
    helper-class="org.ofbiz.entity.datasource.GenericHelperDAO"
    schema-name="OFBIZ"
    field-type-name="derby"
    check-on-start="true"
    add-missing-on-start="true"
    use-pk-constraint-names="false"
    alias-view-columns="false">
    <read-data reader-name="seed"/>
    <read-data reader-name="demo"/>
    <read-data reader-name="ext"/>
    <inline-jdbc
        jdbc-driver="org.apache.derby.jdbc.EmbeddedDriver"
        jdbc-uri="jdbc:derby:ofbiz;create=true"
        jdbc-username="ofbiz"
        jdbc-password="ofbiz"
```

```
isolation-level="ReadCommitted"
pool-minsize="2"
pool-maxsize="20"/>
</datasource>
```

In the datasource `localderby`, the Derby "database instance", so to speak, is in the folder `ofbiz` in the folder `${OFBizInstallFolder}\runtime\data\derby`. That is specified by the string `ofbiz` (part of `jdbc:derby:ofbiz;`) in the attribute `jdbc-uri` of the element `<inline-jdbc>`. To create another database instance, simply create another datasource specifying a different folder, say `jdbc:derby:another`.

The schema name specified in the above datasource is `OFBIZ`. It is not necessary to create another database instance simply to have a separate datasource. Derby supports multiple schemas, so we can have segregated databases using multiple schemas.

However, in the case of Derby, it is often useful to have separate databases in separate database instances so we can back up and restore each one separately. Having two databases within two schemas inside a single database instance will mix the two into the same folder `ofbiz` in the folder `${OFBizInstallFolder}\runtime\data\derby`, making it difficult to backup or restore either one independently of the other.

Entity Delegators

OFBiz uses an "entity delegator" to access the database(s). A delegator in software engineering is like an administrative object that delegates work to other objects (Delegation Pattern). An entity delegator's main purpose is to provide database access methods for the creation, retrieval, update, and deletion of data (CRUD). However, because it is a delegator, it doesn't perform those access tasks itself. Instead, it will determine which datasource to use for accessing which entity groups (explained next) of entities, and then delegate the actual task of accessing to those datasources.

Entity delegators are defined in `${component:entity}\config\entityengine.xml`. An entity delegator is defined by a `<delegator>` element. By default, OFBiz webapps use the delegator named `default`:

```
<delegator name="default" entity-model-reader="main"
           entity-group-reader="main" entity-eca-reader="main"
           distributed-cache-clear-enabled="false">
  <group-map group-name="org.ofbiz" datasource-name="localderby"/>
</delegator>
```

The above entity delegator handles only a single entity group named `org.ofbiz`. The datasource to use for this group is `localderby`. The other attributes of the `<delegator>` element will be discussed later.

Entity Groups

An entity group is a group of entities organized under a group name. The group name is used as a handle for entity delegators to determine which datasource to look into for which entities. Out of the box, only one entity group is ever used by all entity delegators defined for OFBiz: org.ofbiz. All entities in OFBiz are classified under that entity group.

The assigning of entities to entity groups is done in a file named entitygroup.xml in a folder named entitydef inside a component folder. One example is \${component:party}\entitydef\entitygroup.xml. To assign an entity PostalAddress to an entity group org.ofbiz, we would add:

```
<entity-group group="org.ofbiz" entity="PostalAddress"/>
```

Defining Our First Entity

Entities in OFBiz are by convention defined in files named entitymodel.xml inside the same entitydef (entity definitions) folder as the entitygroup.xml file. For example, the entity definitions for the component party are found in the file \${component:party}\entitydef\entitymodel.xml.

An entity is defined by an <entity> element. Let's create our own entity. In the folder \${component:learning}, create a new folder entitydef. In that new folder, create a new file named entitymodel.xml and enter into it this:

```
<?xml version="1.0" encoding="UTF-8"?>
<entitymodel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation=
              "http://www.ofbiz.org/dtds/entitymodel.xsd">

    <entity entity-name="Planet" package-name="org.ofbiz.learning">
        <field name="planetId" type="id-ne"/>
        <field name="planetName" type="name"/>
        <field name="fromDate" type="date-time"/>
        <field name="thruDate" type="date-time"/>
        <prim-key field="planetId"/>
        <index name="PLANET_NAME_IDX" unique="true">
            <index-field name="planetName"/>
        </index>
    </entity>
</entitymodel>
```

Assigning Our Entity to an Entity Group

Now we need to put our entity under the entity group `org.ofbiz`. In the folder `${component:learning}\entitydef`, create a new file `entitygroup.xml` and enter into it this:

```
<?xml version="1.0" encoding="UTF-8"?>
<entitygroup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation=
                  "http://www.ofbiz.org/dtds/entitygroup.xsd">
    <entity-group group="org.ofbiz" entity="Planet"/>
</entitygroup>
```

Loading Our Entity into the Entity Engine

Finally, we need to tell OFBiz to load our entity definition files: `entitymodel.xml` and `entitygroup.xml`. We do that in the file `${component:learning}\ofbiz-component.xml`. Right below the last `<classpath>` element, insert two `<entity-resource>` elements like this:

```
<entity-resource type="model" reader-name="main" loader="main"
                 location="entitydef/entitymodel.xml"/>
<entity-resource type="group" reader-name="main" loader="main"
                 location="entitydef/entitygroup.xml"/>
```

The `<entity-resource>` elements tell OFBiz to load our two entity definition files. Those files are called "resources" in the Java context. A resource is any file containing data that can be read by a Java application (OFBiz in this case). In general, a resource is like database data but cannot be changed at runtime. Whereas database data can be manipulated (by CRUD processes), resource data is hard-coded in physical files (like `entitymodel.xml`) and changes to these physical files will require a restart to take effect.

The attribute `reader-name` tells OFBiz to use an `<entity-model-reader>` named `main`. This is specified in the file `${component:entity}\config\entityengine.xml`. It is not common to use more than one entity model reader or more than one entity delegator, so we shall not cover this in this book. The single entity model reader defined in OFBiz is named `main`.

The attribute `loader` points to a named `<resource-loader>`. Let us now insert a `<resource-loader>` element right above the first `<classpath>` element, like this:

```
<resource-loader name="main" type="component"/>
```

The `<resource-loader>` does not really create or define a loader. Actually, only one loader is present. Different `<resource-loader>` elements with different type attributes cause the single resource loader to function differently. For the above `<resource-loader>` named `main` of type `component`, the resource loader prefixes the component's root path (`${component:learning}` in this case) to the location attribute of an `<entity-resource>` when loading an `<entity-resource>`. For example, `path entitydef\entitymodel.xml` will be prepended to form `${OFBizInstallFolder}\hot-deploy\learning\entitydef\entitymodel.xml`.

In general, the best practice is to use a `<resource-loader>` of type `component`, so that we can keep resource files neatly contained within the component's folder. To illustrate this neat organization, let us consider an alternative. Using a `<resource-loader>` of type `file` instead of `component` will require us to write our `<resource-loader>` like this:

```
<entity-resource type="model" reader-name="main" loader="main"
    location=". ./hot-deploy/learning/entitydef/entitymodel.xml"/>
```

where the `". "` in attribute `location` points to `${OFBizInstallFolder}`.

Seeing Our First Entity

Restart OFBiz to put the new entity into the database. Open up the file `ofbiz.log` and observe the entries regarding the creation of our new entity:

```
2008-04-23 20:22:34,640 (main) [DatabaseUtil.java:853:INFO ] Getting
Table Info From Database
2008-04-23 20:22:36,328 (main) [DatabaseUtil.java:988:INFO ] Getting
Column Info From Database
2008-04-23 20:22:40,343 (main) [DatabaseUtil.java:351:WARN ] Entity
[Planet] has no table in the database
2008-04-23 20:22:42,734 (main) [DatabaseUtil.java:364:INFO ] Created
table [OFBIZ.PLANET]
```

Entities, View Entities, and Extended Entities

To see the entity, navigate to the webtools component's **Entity Data Maintenance** screen by firing an https OFBiz main to webapp webtools and selecting **Entity Data Maintenance** from the list.

Entity Data Maintenance				
Note: Crt :- Create New Reln :- View Relations Fnd :- Find Record All :- Find All Records				
A B C D E F G I J K L M N O P Q R S T U V W X Z				
Entity Name			Entity Name	
AcctgTrans			AcctgTransAndEntries (View Entity)	
AcctgTransAttribute			AcctgTransEntry	
AcctgTransEntryProdSums (View Entity)			AcctgTransEntryType	
AcctgTransType			AcctgTransTypeAttr	
Addendum			AddressMatchMap	
Affiliate			Agreement	
AgreementAttribute			AgreementGeographicalAppl	
AgreementItem			AgreementItemAndPartyAppl (View Entity)	
AgreementItemAndProductAppl (View Entity)			AgreementItemAttribute	
AgreementItemType			AgreementItemTypeAttr	
AgreementPartyAppl			AgreementProductAppl	
AgreementPromoAppl			AgreementRole	
AgreementTerm			AgreementTermAttribute	
AgreementType			AgreementTypeAttr	
AgreementWorkEffortAppl			ApplicationSandbox	

Look the entry for entity **Planet** and click its **Fnd** button (short for "Find").



From here on, we will refer to these screens as the **Entity Data Maintenance** screens. The screen below will be referred to as the entity data maintenance screen for the entity **Planet**.

Find Values

For Entity: Planet

[Back To Entity List](#) [View Relations](#) [Find All](#) [Create New Planet](#)

To find ALL of Entity Planet, leave all entries blank.

Field Name	Primary Key	Field Type	Find
planetId	*	String, VARCHAR(20)	<input type="text"/>
planetName		String, VARCHAR(100)	<input type="text"/>
fromDate		java.sql.Timestamp, TIMESTAMP	<input type="text"/>
thruDate		java.sql.Timestamp, TIMESTAMP	<input type="text"/>
lastUpdatedStamp		java.sql.Timestamp, TIMESTAMP	<input type="text"/>
lastUpdatedTxStamp		java.sql.Timestamp, TIMESTAMP	<input type="text"/>
createdStamp		java.sql.Timestamp, TIMESTAMP	<input type="text"/>
createdTxStamp		java.sql.Timestamp, TIMESTAMP	<input type="text"/>
Find			

[Create New Planet](#)

planetId	planetName	fromDate	thruDate	lastUpdatedStamp	lastUpdatedTxStamp	createdStamp	createdTxStamp
----------	------------	----------	----------	------------------	--------------------	--------------	----------------

No Records found for Entity Planet.

[Create New Planet](#)

Using Our First Entity

In the section called *Changing the Looks* in Chapter 2, we inserted a form field planet into the **Edit Contact Information** screen. Fire an https OFBiz request to webapp partymgr and bring up any party, say last name **Researcher** and first name **OFBiz**. Enter this party profile and edit (update) one of the existing Postal Addresses. Create a new one if there are none listed. The familiar form field planet should show up. It is currently a text field, for free-form text entry.

To Name	<input type="text" value="Gift Recipient"/>
Attention Name	<input type="text"/>
Address Line 1	<input type="text" value="Somewhere"/>
Address Line 2	<input type="text"/>
City	<input type="text" value="A City"/>
State/Province	<input type="text" value="AK"/> <input type="button" value="▼"/>
Zip/Postal Code	<input type="text" value="123456"/>
Country	<input type="text" value="United States"/> <input type="button" value="▼"/>
Planet	<input type="text" value="Earth"/>
Allow Solicitation?	<input type="button" value="▼"/>
Go Back Save	

Creating a Drop-Down

Instead of a free-form text field, we will provide a drop-down list populated with values from the entity `Planet` that we just created. In the file `component:learning\widget\partymgr\OurPartyScreens.xml` create a new screen widget called `editcontactmech.planet.dropdown`:

```
<screen name="editcontactmech.planet.dropdown">
<section>
    <actions>
        <entity-one entity-name="Planet" value-name="planet">
            <field-map field-name="planetId"
                env-name="mechMap.postalAddress.planet"/>
        </entity-one>
        <entity-condition entity-name="Planet" filter-by-date="true"
            list-name="planets">
            <order-by field-name="planetName"/>
        </entity-condition>
    </actions>
    <widgets>
        <platform-specific><html>
            <html-template location="component://learning/webapp/partymgr/
                party/editcontactmech.planet.dropdown.ftl"/>
        </html></platform-specific>
    </widgets>
</section>
</screen>
```

The `<entity-condition>` element queries the database and pulls down all records in the entity `Planet`. The list of records is stored in the list variable `planets`. The sub-element `<order-by>` tells the query to return the results sorted by the field `Planet.planetName`. The default sort order is ascending alphabetically. To get a descending order, set the attribute `field-name` to `planetName DESC`.

The `filter-by-date` attribute specifies that the query must filter out those records that are not active now. Inactive records have the fields `fromDate` and `thruDate` (both date-time fields) that don't frame the current time. Recall that the entity `Planet` has those two fields. This is a neat feature in OFBiz that provides for easy audit trail mechanism. Rather than deleting a record, it is merely expired. An empty `fromDate` is deemed to stretch to forever into the past; an empty `thruDate` covers forever into the future.

The `<entity-one>` element retrieves the record from entity `Planet` that has the field `Planet.planetId` with a value matching the postal address `PostalAddress.planet` field. If found, the record is placed into the variable `planet`. Note how the entity `PostalAddress` is related to the entity `Planet` via the field `PostalAddress.planet`. We will discuss relationships between entities later.

These elements in the `<actions>` element are actually Minilang—the OFBiz programming language. It will be discussed later in this book.

Now, we create the corresponding "html template". In the folder `${component:learning}\webapp\partymgr\party` create a new file called `editcontactmech.planet.dropdown.ftl` and enter into it this:

```

<tr>
  <td class="label">Planet</td>
  <td>
    <#assign fieldName = "planet"/>
    <select name="${fieldName}">
      <#if (planet)?>
        <option value="${planet.planetId}" />${(planet.planetName) ! "" }
      </option>
    </#if>
    <option></option>
    <#if request.getParameter("${fieldName}")??>
      <#assign requestPlanet = request.getParameter("${fieldName}") />
    </#if>
    <#list planets as planet>
      <option value="${planet.planetId}"
        <#if requestPlanet?? && requestPlanet ==
          planet.planetId>selected</#if>
        ${planet.planetName}
      </option>
    </#list>
  </select>
</td>
</tr>

```

Lastly, we point `editcontactmech.ftl` to include the new screen widget. Edit `${webapp:party}\party\editcontactmech.ftl` and replace (around line 185):

```

${screens.render("component://learning/widget/partymgr/
  OurPartyScreens.xml#editcontactmech.extend") }

```

with this:

```

${screens.render("component://learning/widget/partymgr/
  OurPartyScreens.xml#editcontactmech.planet.dropdown") }

```

Go back to the **Edit Contact Information** screen to see the new drop-down field **Planet**.

To Name	Gift Recipient
Attention Name	
Address Line 1	Somewhere
Address Line 2	
City	A City
State/Province	AK
Zip/Postal Code	123456
Country	United States
Planet	<input type="button" value="▼"/>
Allow Solicitation?	<input type="button" value="▼"/>
Go Back Save	

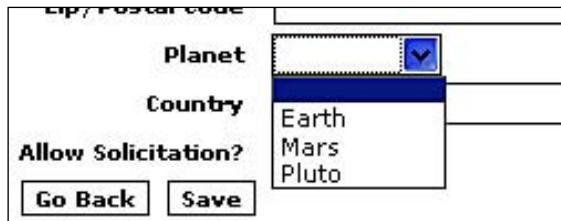
If we were building the equivalent of the above `editcontactmech.planet.dropdown.ftl` with a form widget, we would have done:

```
<form name="PlanetDropDown" type="single" skip-start="true" skip-end="true">
    <field name="planetId" title="Planet"
        map-name="mechMap.postalAddress">
        <drop-down allow-empty="true" current="selected">
            <entity-options description="${planetName}">
                entity-name="Planet"
                key-field-name="planetId"
                filter-by-date="true">
                    <entity-order-by field-name="planetName"/>
            </entity-options>
        </drop-down>
    </field>
</form>
```

Populating the Drop-Down

The drop-down is still empty. It is pulling its options population from the database entity **Planet**. Go to the entity data maintenance for the entity **Planet**. Click **Create New Planet**, then click **Edit**. Enter for fields **planetId** and **planetName** values of **MARS** and **Mars**, respectively. Create a second record with the values **EARTH** and **Earth**. Create a 3rd record with the values **PLUTO** and **Pluto**.

Go back to the **Edit Postal Address** screen to see the three values in the drop-down field **Planet**.



Expiring a Value

Suppose we now decide that Pluto is too far away to colonize. Rather than removing the planet from the database altogether, we expire it. That way, we can always remember that we had a value `Pluto` before. To expire Pluto, enter for its field **thruDate** a value that is before now, say last month.

Go back to the **Edit Contact Information** screen to see that there are two values left. Pluto has been expired.

Un-Expiring a Value

To un-expire a value, we can set in field **thruDate** a time far in the future. However, we would have to come back to un-expire the value again once that future time has come. The correct way to un-expire a record for good (until further notice) is to set the **thruDate** field to null (that means "empty" or "no value" in computing terminology).

Delete the planet Pluto's value for **thruDate** and leaving it blank, click **Update** to un-expire this record.

Anatomy of an <entity> Element

The attribute name of an `<entity>` element is required; it uniquely identifies the entity in the entire Entity Engine. There cannot be entities with similar names, even if they are placed into different entity groups or packages (explained next).

The attribute `package-name` simply serves as a taxonomical tool, allowing OFBiz to deal with entities, for example, under different packages differently. So far, it is only used by the Entity Engine's data synchronization tool, which is related to Online Analytical Processing (OLAP) and involves multiple database machines (one active and one for OLAP), something we won't be covering in this book.

An `<entity>` element can contain `<field>` elements, `<prim-key>` elements, `<relation>` elements, and `<index>` elements, in that order. There must be at least one `<field>` element.

Let's take another look at our `Planet` entity that we defined in the `entitymodel.xml` file:

```
<entity entity-name="Planet" package-name="org.ofbiz.learning">
    <field name="planetId" type="id-ne"/>
    <field name="planetName" type="name"/>
    <field name="fromDate" type="date-time"/>
    <field name="thruDate" type="date-time"/>
    <prim-key field="planetId"/>
    <index name="PLANET_NAME_IDX" unique="true">
        <index-field name="planetName"/>
    </index>
</entity>
```

The `<field>` Element

The `<field>` elements define fields in an entity. Each `<field>` element must at least have the attributes `name` and `type` defined. Obviously, the `name` attribute is required to provide a unique reference or handle on the field. The `type` attribute determines the data type that the field will and must contain.

Database Independence

Different databases have different data types, just like different languages have different vocabularies. But just as languages can have equivalent counter-parts, different databases also often have equivalent data types between them. OFBiz achieves database independence (ability to couple with different databases at the flip of a switch) by defining standard "OFBiz data types"—types that mean something to OFBiz. For each of these OFBiz data types, OFBiz has a mapping to an equivalent database specific data type. This way, OFBiz code needs only deal with a single set of data types—OFBiz data types. OFBiz is in charge of translating its own data types into database-specific ones.

The data type mappings are in the folder `${component:entity}\fieldtype`. Supported databases are Derby, PostgreSQL, MySQL, MSSQL, Oracle, MaxDB, and many others. The folder contains one mapping file `fieldtype<database-name>.xml` for each database OFBiz supports (for example `fieldtypederby.xml`). Each mapping file is loaded in file `${component:entity}\config` by a `<field-type>` element.

The "type" Attribute

Each `<field>` element must have a `type` attribute specified. The `type` attribute must have a value of one of the OFBiz data types. The full list of data types is listed in any of the mapping files, such as `fieldtypederby.xml`.

In that mapping file, each data type is defined by a `<field-type-def>` element. Here is one example:

```
<field-type-def type="date-time" sql-type="TIMESTAMP"
                 java-type="java.sql.Timestamp">
```

The data type name is in attribute `type`. The attribute `sql-type` is the RDBMS-specific data type that this OFBiz data type maps to. The attribute `java-type` tells OFBiz which Java type to use to handle the data type.

The `<prim-key>` Element

Every record in an entity needs a unique identity or signature so that it can be referenced. This unique identity in database terms is known as a **primary key**. The primary key of an entity is defined by its `<prim-key>` elements.

There can be multiple `<prim-key>` elements. Multiple elements do not mean there are multiple ways to identify a record. Some entities may require more details to uniquely identify their records. Using the namespace analogy we used before, two people named "Peter Parker" will have the same values for the key that is their names. An additional key will be required to uniquely identify them, say a key that is their address of residence. If they happen to live in the same house, yet another key will be needed! Some countries require a unique identity number for their citizens.

A more realistic example in the computing context is an entity that represents a relationship between two other entities, say between `Party` (a person or group) and `PostalAddress`. We have an entity `PartyContactMech` that records which person lives where. The primary key for this entity would be two fields—the unique ID for `Party` and that for `PostalAddress`. A schema designed like that will mean a person can live in more than one location, and each location can house more than one person.

The `<relation>` Element

Relationships between entities are defined by `<relation>` elements, and are called **entity relations**. Using the `PartyContactMech` entity as an example, we know that the field `partyId` relates to the `Party` entity. We could therefore place a foreign key relationship on the `PartyContactMech.partyId` field and the `Party.partyId` field. We can now ensure that all records in the `PartyContactMech` entity are related to a party in the `Party` entity. Therefore the `PartyContactMech` entity cannot contain information on a party that does not have a record in the `Party` entity.

Creating the Field that Participates in a Relation

In the file \${component:party}\entitydef\entitmodel.xml, edit entity `PostalAddress` and insert below the existing field `planet` (which we added in the section called *Changing the Data* in Chapter 2) at line 962:

```
<field name="planet" type="name"></field>
```

a new field "planetId" like this:

```
<field name="planetId" type="id"></field>
```

Right now, this field `planetId` is just a field. We need to create a relation that uses this field.

Creating the Relation

We now create a relation between the entities `PostalAddress` and `Planet`. Edit the entity `PostalAddress` and insert right below the `<prim-key>` element at line 967:

```
<prim-key field="contactMechId"/>
```

a new `<relation>` element like this:

```
<relation type="one" fk-name="POST_ADDR_PLANET"
          rel-entity-name="Planet">
  <key-map field-name="planetId"/>
</relation>
```

The **foreign key** that links the entity `PostalAddress` to the entity `Planet` is the field `PostalAddress.planetId`, as specified by the `<key-map>` sub-element. By default, the `<key-map>` links the related entities via fields with the same name as specified in the attribute `field-name` (`PostalAddress.planetId` and `Planet.planetId` in this case). It is possible to point the field `PostalAddress.planetId` to some other field by specifying an additional attribute `rel-field-name` with a value of say `someOtherField`.

Each `<relation>` element must have at least one `<key-map>` sub-element.

The `type` attribute specifies the type of relation. In our new relation, each `PostalAddress` record points to exactly one `Planet` record. There can be other types of relations, which we will discuss later.

The `fk-name` attribute specifies the name of this foreign key (the field `PostalAddress.planetId`), and can also be considered the name of this relation. Although OFBiz can automatically generate a foreign key name if none is specified, the generated name might be too long for some databases (some have a maximum length as short as 18 characters). It is best to have a consistent convention in naming relations. In our relation, the foreign key name `POST_ADDR_PLANET` suggests that it is a relation from a postal address to a planet.

The `rel-entity-name` attribute points to the other end of this relation, the entity that `PostalAddress` is related to. Restart OFBiz to let OFBiz pick up the new `<relation>` element in the XML definition file `${component:party}\entitydef\entitymodel.xml`.

Return to the OFBiz entity data maintenance screen for `PostalAddress` and select **View Relations**.

<code>PostalCode</code>	<code>Geo</code>	<code>one</code>	<code>POST_ADDR_PC GEO</code>	<code>postalCodeGeoId -> geoId</code>
	<code>Planet</code>	<code>one</code>	<code>POST_ADDR_PLANET</code>	<code>planetId -> planetid</code>
	<code>Billing Account</code>	<code>many</code>		<code>contactMechId -> contactMechId</code>

Accessing the Relation

Recall that we previously used the field `PostalAddress.planet` as the unofficial foreign key or link to the field `Planet.planetId`. In the following code, the variable `mechMap.postalAddress` is a `PostalAddress` record, so `mechMap.postalAddress.planet` will refer to the field `planet` in the entity `PostalAddress`. Look in the screen widget `editcontactmech.planet.dropdown` for this:

```
<entity-one entity-name="Planet" value-name="planet">
  <field-map field-name="planetId"
    env-name="mechMap.postalAddress.planet"/>
</entity-one>
```

However, the relation we just created uses the field `PostalAddress.planetId` as the foreign key. Do we need to change `mechMap.postalAddress.planet` to `mechMap.postalAddress.planetId`? With the OFBiz entity relation framework, we just need to tell OFBiz that we want the relation `Planet` of the entity `PostalAddress`. We don't need to bother with finding out exactly which fields are linked up, `planet` or `planetId`.

In the screen `editcontactmech.planet.dropdown`, replace the `<entity-one>` element with this:

```
<get-related-one value-name="mechMap.postalAddress"
  relation-name="Planet" to-value-name="planet"/>
```

The above `<get-related-one>` element tells OFBiz we want to get the relation named `Planet` for the `PostalAddress` record we are holding in the variable `mechMap.postalAddress`. The related `Planet` record is placed into the variable `planet`. Test this and see it work.

The name of a relation is typically the name of the related entity, as defined by the attribute `rel-entity-name` of the element `<relation>`. Sometimes, there may be more than one relation to the same entity, such as when country codes and state codes are all stored in the entity `Geo` (for "Geographical Location"). In this case, the attribute `title` will distinguish between such relations by prefixing itself to the attribute `rel-entity-name` to form the relation name. In the entity `PostalAddress`, for example, there are two relations named `CountryGeo` and `StateProvinceGeo` to the entity `Geo`:

```
<relation type="one" fk-name="POST_ADDR_CGEO" title="Country"
          rel-entity-name="Geo">
    <key-map field-name="countryGeoId" rel-field-name="geoId"/>
</relation>
<relation type="one" fk-name="POST_ADDR_SPGEO" title="StateProvince"
          rel-entity-name="Geo">
    <key-map field-name="stateProvinceGeoId" rel-field-name="geoId"/>
</relation>
```

Since we are no longer using the field `PostalAddress.planet`, but using the field `planetId` instead, we should change our form field accordingly. Edit the file `editcontactmech.planet.dropdown.ftl` and replace line 3:

```
<#assign fieldName = "planet"/>
```

with:

```
<#assign fieldName = "planetId"/>
```

And finally we must change out the `postalAddressAdvisory` method in package `org.ofbiz.learning.LearningEvents` to this:

```
public static final String module = LearningEvents.class.getName();
public static String postalAddressAdvisory(HttpServletRequest
request, HttpServletResponse response){
    String partyId = request.getParameter("partyId");
    Map mechMap = new HashMap();
    ContactMechWorker.getContactMechAndRelated(request, partyId,
                                                mechMap);
    GenericValue postalAddress =
        (GenericValue)mechMap.get("postalAddress");
    if (postalAddress == null) return "notMars";
```

```

GenericValue planetGV = null;
try{
    planetGV = postalAddress.getRelatedOne("Planet");
} catch(GenericEntityException e){
    Debug.logError(e, "An Error Occurred", module);
}
String planet = (planetGV != null) ? planetGV.
getString("planetId") : null;
if (planet == null || !planet.equalsIgnoreCase("Mars")) return
"notMars";

return "isMars";
}

```

There are a number of new concepts in this code. Do not worry if they are not fully understood at the moment, they will all be explained later in the book. The following import statements must be added to the class:

```

import org.ofbiz.entity.GenericEntityException;
import org.ofbiz.entity.GenericValue;

import org.ofbiz.base.util.Debug;

```

The `GenericValue` is a Java class used by OFBiz as a holder for any arbitrary database record. A single `GenericValue` represents a single record. OFBiz uses Apache Log4J to produce the log files and the `Debug` class is a utility class that writes to the logs depending on the settings in the `debug.properties` file in the `${OFBizInstallFolder}\framework\base\config` folder.



Don't forget to shutdown OFBiz and recompile the learning component!



Updating the Database

Although OFBiz now recognizes that the relation has been added to the entity `PostalAddress`, the database is still unaware of it. As mentioned in the section called *Updating the Database* in Chapter 2, we merely have to restart OFBiz whenever we add new entities or entity fields. However, updating the database with foreign keys isn't so trivial.

There are some quirks here. If an entity has not yet been created, OFBiz will create it as well as any foreign keys in it. In our case, the entity `PostalAddress` already existed when we added the new relation. Deleting the entity is not a good option because we will lose all the existing data in there.

For now, we want to confirm that the foreign key name `POST_ADDR_PLANET` really does not yet exist in the database. Go to the entity data maintenance screen for the entity `PostalAddress` to look at the `PostalAddress` records.

Click on **Find All** to pull down around 13 records (more if you had played with OFBiz postal addresses a great deal). Pick the first record, say with `contactMechId` of **9000**, and click on its **View** button. Click **Edit**, and enter into the field `planetId` a value that does not exist in the entity `Planet`, say **VENUS**. That is, enter any value that is not **MARS**, **EARTH**, or **PLUTO**. Click on **Update**. You'll see that the update was successful. This means that the foreign key `POST_ADDR_PLANET` is not in the database.

Change the value of the field `planetId` back to null (empty) or to any values existing in the entity `Planet.planetId`, such as `EARTH`. Having a `PostalAddress` record that points to a non-existent `Planet` record will prevent us from creating the foreign key in the database.

One of the main benefits of foreign keys is that most (if not all) databases help maintain "data integrity" with them. The database makes sure that all foreign keys point to something that exists, so that all foreign keys are meaningful. It is meaningless for foreign keys to point to a non-existent record, as well as misleading.

The `<index>` Element

The records in a database entity can possibly be stored in some random order, and arbitrary locations on the hard disk. Just like a directory or index of "people-to-address" helps us locate people in the real world scattered into various locations, an index in a database speeds up the retrieval of records.

Besides serving as a performance tool enhancing data retrieval speeds, indexes also allow us to enforce certain constraints, such as what we have in the entity `Planet`: no two planets can have the same value for the field `planetName`.

```
<index name="PLANET_NAME_IDX" unique="true">
    <index-field name="planetName" />
</index>
```

In each entity, indexes can be defined on any field. It is defined by an `<index>` element like this:

```
<index name="POSTAL_CODE_IDX">
    <index-field name="postalCode" />
</index>
```

It is likely that there is a common need to search postal addresses by postal code, hence the index to speed up data retrieval. Other fields in the entity `PostalAddress` that have indexes are `address1`, `address2`, and `city`.

Just as a foreign key has a name (defined in the attribute `fk-name` of a `<relation>` element), an index also has a name. It is defined in the attribute `name`.

The attribute `unique`, if `true`, will enforce a unique constraint over the fields defined under the index. Note that an index defined over multiple fields will only have its unique constraint violated if there are similar combinations of values of those fields. This is similar to a prior example in this chapter, where we explored primary keys defined over multiple fields (`Person` and `PostalAddress`). Primary keys inherently have a unique constraint.

The sub-element `<index-field>` specifies which entity field this index is defined on. In the above example, the index `POSTAL_CODE_IDX` is defined on the field `PostalAddress.postalCode`. Just like primary keys defined over multiple fields, indexes can also be defined over multiple fields.

It may seem tempting to go ahead and add an index on every field that is being looked up, or even those that may be looked up in the future, however careful consideration should be given as to which fields are indexed. If an index was placed on every field, the overhead of maintaining the indexes would far outweigh the benefits.

Indexes on Foreign Keys

It is often a good idea to index the foreign keys. Some databases like MySQL automatically create an index on foreign keys because:

"[A database] requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. The index on the foreign key is created automatically."

(This excerpt has been taken from the MySQL documentation on foreign keys <http://dev.mysql.com/doc/refman/5.0/en/innodb-foreign-key-constraints.html>)

A "foreign key check" is a check by the database that all foreign keys refer to existing records. For example, if the entity `Planet` does not have a record with a `planetId` of `VENUS`, then there should be no records in `PostalAddress` that have a `planetId` with the value of `VENUS`. The database goes through all the records with the entity `PostalAddress` to do this check. Rather than hunting all over the hard disk for all the records of `PostalAddress`, a convenient index on the foreign key `PostalAddress.planetId` can quickly call up every record (that has a non-empty `planetId`) for checking.

OFBiz will automatically create indexes on foreign keys if the datasource has the attribute `use-foreign-key-indices` set to `true`, which is the case by default. The datasource `localderby` does not specify this attribute, so it is `true` by default.

A good practice is to avoid specifying indexes on foreign keys when developing in OFBiz. It will only add clutter, since OFBiz can be centrally configured (at the datasource definition) to automatically create indexes on foreign keys. Since many databases do create implicit indexes on foreign keys, you may also choose to turn off the `use-foreign-key-indices` so that indexes are not created doubly over the same fields.

Relation Types

There are primarily two types of entity relations: one-to-one and one-to-many. They are denoted by `<relation type="one">` and `<relation type="many">` elements, respectively. Entity relations are always defined as between two entities.

The first thing to note is the left side of the `-to-` where we define an entity relation. Of the many fields in a person's particulars, one of them, say the residence address field, may point to a postal address; the person record is the **source** of the relation. The postal address record minds its own business, and has no explicit link (no foreign key) pointing to the person; it is the **destination** of the relation. That is a good rule of thumb to keep in mind when deciding where to define entity relations.

We define entity relations on the source or referencing entity, with the relations pointing to the destination or referenced entity. A relation defined on an entity can be expressed like "points to related entity". A one-to-one relation means 'points to one record of related entity'; a one-to-many relation means 'points to many records of related entity'.

One-to-One Relations

A type "one" entity relation in OFBiz is defined by a `<relation type="one">` element. It creates a foreign key with one or more of the entity's fields, as specified by the one or more `<key-map>` sub-elements.

The foreign key must contain the same fields in the same order as the primary key fields in the referenced entity. Every foreign key must either reference a unique record in the referenced entity, or be null and reference nothing at all.

One-to-Many Relations

A type "many" relation is defined by a `<relation type="many">` element. No foreign key is created.

Inverse of One-to-One Relations

One-to-Many relations are often merely the inverse of One-to-One relations. It's like peering through the other end of the telescope. Say there are several records in the entity `PostalAddress` referring to a single record in `Planet`, which means several postal addresses are on the same planet. To each postal address, it is one postal address record referring to one planet record, hence the one-to-one relation. To the planet record, on the other hand, it sees many postal address records referring to one planet record that is itself.

From the perspective of the entity `PostalAddress`, it has a one `PostalAddress` record to one `Planet` record relation, a one-to-one relation with entity `Planet`. To the entity `Planet`, it has a one `Planet` record to many `PostalAddress` records relation, a one-to-many relation.

There is no need to define type "many" relations in such cases because one is implicitly created for every type "one" relation.

Since a pair of corresponding relations: type "one" and type "many", can be succinctly and sufficiently defined by a single type "one" relation, OFBiz does not create foreign keys for type "many" relations. It doesn't make sense to do so anyway, because a foreign key must point clearly to a unique referenced record rather than point ambiguously to multiple records. In fact, in many databases, there is no such thing as a foreign key for a one-to-many relation (some will argue that this should be the correct behavior for all databases).

To see an implicit type "many" relation in action, let's look at the entity `Planet`. the Entity `Planet` does not specify any relations at all, yet an implicit relation named `PostalAddress` does exist. To see it in action, create a new screen widget `ShowPlanetManyRelation`. In the folder `${component:learning}\widget\learning`, insert into the file `LearningScreens.xml` this:

```
<screen name="ShowPlanetManyRelation">
<section>
<actions>
    <!-- First, get all planets. -->
    <entity-condition entity-name="Planet" filter-by-date="true"
                      list-name="planets">
        <order-by field-name="planetName"/>
    </entity-condition>
</actions>
<widgets>
    <!-- For each planet, show addresses -->
    <iterate-section entry-name="planet" list-name="planets">
        <section>
            <actions>
```

```
<get-related value-name="planet"
             relation-name="PostalAddress"
             list-name="planetAddresses"/>
</actions>
<widgets>
  <platform-specific><html>
    <html-template location="component://learning/webapp/
                           learning/planetAddresses.ftl"/>
  </html></platform-specific>
</widgets>
</section>
</iterate-section>
</widgets>
</section>
</screen>
```

In the folder \${webapp:learning}, create a new file planetAddresses.ftl and enter into it this:

```
<b>Planet: <i>${planet.planetName}</i></b>
<br/>
<table border="1">
<tr>
<td>contactMechId</td>
<td>Address</td>
<td>City</td>
<td>Postal Code</td>
</tr>
<#list planetAddresses as address>
<tr>
<td>${address.contactMechId}</td>
<td>${address.address1!""} ${address.address2!""}</td>
<td>${address.city!""}</td>
<td>${address.postalCode!""}</td>
</tr>
</#list>
</table>
```

In the folder \${webapp:learning}\WEB-INF, insert into the file controller.xml a new request map:

```
<request-map uri="ShowPlanetManyRelation">
<response name="success" type="view" value="ShowPlanetManyRelation"/>
</request-map>
```

and a new view map:

```
<view-map name="ShowPlanetManyRelation" type="screen"
          page="component://learning/widget/learning/
                LearningScreens.xml#ShowPlanetManyRelation"/>
```

From the data maintenance screen for the entity `PostalAddress`, edit a few postal addresses to link them (via the field `planetId`) to any of the three planets: EARTH, MARS, and PLUTO. Fire an http OFBiz request `ShowPlanetManyRelation` to webapp learning to see the addresses listed according to the planets they are on.

Planet: Earth			
<code>contactMechId</code>	<code>Address</code>	<code>City</code>	<code>Postal Code</code>
9014	1234 Commerce Way	Los Angeles	90010
Planet: Mars			
<code>contactMechId</code>	<code>Address</code>	<code>City</code>	<code>Postal Code</code>
CA_BOE_0	P O BOX 942879	Sacramento	94279
NY_DTF_0	JAF Building PO BOX 1205	New York	11016
9010	2004 Factory Blvd	Orem	84057
Planet: Pluto			
<code>contactMechId</code>	<code>Address</code>	<code>City</code>	<code>Postal Code</code>
9000	2003 Open Blvd	Open City	999999
9200	2003 Open Blvd	Orem	84058

Defining an explicit type "many" relation in the entity `Planet` is not necessary. In fact, OFBiz will even warn against that. In the folder `${component:learning}\entitydef`, edit the file `entitymodel.xml` to add to the entity `Planet` this:

```
<relation type="many" rel-entity-name="PostalAddress">
    <key-map field-name="planetId"/>
</relation>
```

Restart OFBiz. OFBiz will refuse to create the explicitly defined relation because it already exists implicitly. In the folder `${OFBizInstallFolder}\runtime\logs`, open file `ofbiz.log` and find the following:

```
Entity [org.ofbiz.learning:Planet] already has identical relationship
to entity [PostalAddress] title [] ; would auto-create: type [many] and
fields [planetId]
```

One-to-One Relations with No Foreign Keys

These relations are defined by `<relation type="one-nofk">` elements. They are legitimate type "one" relations, but without the accompanying foreign keys. They are not pseudo type "one" relations implemented with type "many" relations. Like type "one" relations, they must also point to the primary key in the referenced entity.

They are mostly used for archived records. Say we have a list of both local and out of town wedding guests who have attended a wedding in a small town. This attendance list will have a type "one-nofk" relation to the local town person's registry. Because this attendance list is merely an archived log, we don't want to have a foreign key created between it and the town person's registry.

View Entities

Generally speaking, a view entity is a collection of participating or member entities joined together by a number of relations. This collection forms a structure, where every member entity in a view entity is connected to the overall structure, no entity is left out. In short, a view entity provides a single view of a group of interconnected entities. This is very convenient for performing complex database queries that need to take such high-level views of data.

Technically speaking, a view entity could possibly be a complex hierarchy of relations between a number of member entities. The term **hierarchy** is used because the structure described by a view entity is not cyclical but is strictly top-down. This does not mean that a member entity will not be referenced more than once. A member entity can even be referenced in more than one level, say at a higher level and then again at a lower level, in effect giving the impression that the structure described is cyclical.

When we say hierarchy here, we are not talking about the member entities themselves, but about the roles instead. Each participating entity participates in a view entity in a certain capacity or role. Let's look at a simple example, a view entity containing two participating entities `PersonL` and `HotelL`:

```
<view-entity entity-name="GuestHotelOwnerViewL"
            package-name="org.ofbiz.learning">
  <member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
  <member-entity entity-alias="Hotel" entity-name="HotelL"/>
  <member-entity entity-alias="HotelOwner" entity-name="PersonL"/>
  <view-link entity-alias="HotelGuest" rel-entity-alias="Hotel">
    <key-map field-name="housedAt" rel-field-name="id"/>
  </view-link>
  <view-link entity-alias="Hotel" rel-entity-alias="HotelOwner">
    <key-map field-name="ownedBy" rel-field-name="id"/>
  </view-link>
</view-entity>
```

We use entity names `PersonL` and `HotelL` because the OFBiz base framework already has an entity named `Person`. The `L` is just an arbitrary naming convention we use here, which denotes our "learning" application.

A person can own one or more hotels, and each hotel can have one or more guests. A hotel owner and a hotel guest are both `PersonL` records. That would mean we have a view entity where `PersonL` is connected to `HotelL` is connected to `PersonL`. This seems cyclical. However, when we think in terms of roles, we get a hierarchy—a hotel owner owns one or more hotels which can house one or more hotel guests.

When we say top-down hierarchy, we take the first referencing entity as the "top" or first level. In the above example, that would be a hotel guest. Each hotel guest is housed at one hotel, each hotel is owned by one hotel owner. Each relation in a view entity, defined by a <view-link>, is usually like a type "one" <relation>, though it doesn't have to be.

Anatomy of a <view-entity>

A view entity is made up of four parts:

- the member entities
- the fields (just like fields in an entity)
- the view links (the relations that connect member entities together)
- the relations (just like the relations an entity has with other entities)

The <member-entity> Element

Usually, a view entity has two or more member entities. Each entity's participation or membership in a view entity is defined by a <member-entity> element. A sample is given here:

```
<member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
```

Each member entity, say PersonL, must participate with a specified role, that role being defined by the attribute entity-alias. In the underlying RDBMS, what really happens is that an alias is created for the entity PersonL, hence the attribute name entity-alias. Since an alias is the name that a view entity uses to uniquely reference each participating entity, each alias name must be unique within the view entity. That is, we cannot have:

```
<member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
<member-entity entity-alias="HotelGuest" entity-name="Hotell"/>
```

An entity can participate in a view entity more than once, each time with a different role. For example, the entity PersonL can be a hotel guest as well as a hotel owner:

```
<member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
<member-entity entity-alias="HotelOwner" entity-name="PersonL"/>
```

As discussed, a view entity describes a hierarchy of member entities. By convention, member entities are listed in order according to the hierarchy described. The hierarchy is usually a chain of type "one" relations. For example, if a hotel guest is housed at one hotel, and a hotel is owned by one hotel owner, we will have:

```
<member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
<member-entity entity-alias="Hotel" entity-name="HotellL"/>
<member-entity entity-alias="HotelOwner" entity-name="PersonL"/>
```

A view entity can have one member entity. Although it seems unnatural to have a view entity of just one member entity, this type of view entity is often employed to confine access to a subset of the entity's fields. Say we want to confine a particular application to seeing only a subset of fields in an entity, we would create a view entity with such a subset published. There will be more on this later.

Note that a hierarchy is like a tree, not a linear chain. The examples in this chapter just happen to be a linear chain.

The <alias> and <alias-all> Elements

Like an entity, a view entity has fields. In this respect, think of a view entity like you would a normal entity. In fact, a view entity is accessed or queried in the exact same way an entity is. A view entity GuestHotelOwnerView can have fields like guestFirstName and hotelName, which are referred to as fields

GuestHotelOwnerView.guestFirstName and GuestHotelOwnerView.hotelName.

The <alias> Element

To define a view entity field, we use an <alias> element. Like an <entity-alias>, we're dealing with aliases here. In the folder \${component:learning}\entitydef, edit the file entitymodel.xml to add a new entity PersonL:

```
<entity entity-name="PersonL" package-name="org.ofbiz.learning">
  <field name="personId" type="id-ne"></field>
  <field name="firstName" type="name"></field>
  <field name="lastName" type="name"></field>
  <field name="housedAt" type="id"></field>
  <field name="someExtraDetails" type="description"></field>
  <prim-key field="personId"/>
  <relation type="one-nofk" fk-name="PERSON_HOTEL"
            rel-entity-name="HotellL">
    <key-map field-name="housedAt" rel-field-name="hotelId"/>
  </relation>
</entity>
```

and a new entity "HotelL":

```
<entity entity-name="HotelL" package-name="org.ofbiz.learning">
  <field name="hotelId" type="id-ne"></field>
  <field name="hotelName" type="name"></field>
  <field name="ownedBy" type="id"></field>
  <field name="postalAddressId" type="id"></field>
  <prim-key field="hotelId"/>
  <relation type="one-nofk" fk-name="HOTEL_OWNER"
    rel-entity-name="PersonL">
    <key-map field-name="ownedBy" rel-field-name="personId"/>
  </relation>
</entity>
```

Say we want a view entity with five fields: guest name (first and last), hotel name, and hotel owner (first and last name). In `entitymodel.xml`, add a new view entity `GuestHotelOwnerViewL`:

```
<view-entity entity-name="GuestHotelOwnerViewL"
  package-name="org.ofbiz.learning">
  <member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
  <member-entity entity-alias="Hotel" entity-name="HotelL"/>
  <member-entity entity-alias="HotelOwner" entity-name="PersonL"/>
  <alias entity-alias="HotelGuest" name="guestFirstName"
    field="firstName"/>
  <alias entity-alias="HotelGuest" name="guestLastName"
    field="lastName"/>
  <alias entity-alias="Hotel" name="hotelName" field="hotelName"/>
  <alias entity-alias="HotelOwner" name="ownerFirstName"
    field="firstName"/>
  <alias entity-alias="HotelOwner" name="ownerLastName"
    field="lastName"/>
  <view-link entity-alias="HotelGuest" rel-entity-alias="Hotel">
    <key-map field-name="housedAt" rel-field-name="hotelId"/>
  </view-link>
  <view-link entity-alias="Hotel" rel-entity-alias="HotelOwner">
    <key-map field-name="ownedBy" rel-field-name="personId"/>
  </view-link>
</view-entity>
```

Entries must also be added for these entities into the `entitygroup.xml` file in the same folder:

```
<entity-group group="org.ofbiz" entity="PersonL"/>
<entity-group group="org.ofbiz" entity="HotelL"/>
<entity-group group="org.ofbiz" entity="GuestHotelOwnerViewL"/>
```

Note that, currently, even view entities must be assigned to a group. The above view entity presents a neat view of two entities in three roles, each record with fields guestFirstName, guestLastName, hotelName, ownerFirstName, and ownerLastName.

Go to the data maintenance screen for the entity HotelL and enter two HotelL records like this:

hotelId	hotelName	ownedBy	postalAddressId
1	Brad's Hotel	1	9014
2	Hank's Hotel	3	9300

Take note of the postalAddressId field. We are going to borrow some existing address records for our Hotel to save us from adding new ones. This makes Brad's hotel located in Los Angeles and Hank's hotel located in New York.

Go to the data maintenance screen for the entity PersonL and enter three PersonL records like this:

personId	firstName	lastName	housedAt	someExtraDetails
1	Brad	Pitt	2	
2	Julia	Roberts	1	
3	Tom	Hanks		

Go to the data maintenance screen for view entity GuestHotelOwnerViewL and click on **Find All** to see all records in the view:

guestFirstName	guestLastName	hotelName	ownerFirstName	ownerLastName
Julia	Roberts	Brad's Hotel	Brad	Pitt
Brad	Pitt	Hank's Hotel	Tom	Hanks

Seeing the SQL Equivalent

It may be simpler to see what is happening here by examining the standard SQL equivalent.

```
SELECT HotelGuest.firstName as guestFirstName,  
       HotelGuest.lastName as guestLastName, Hotel.hotelName,  
       HotelOwner.firstName as ownerFirstName,
```

```
    HotelOwner.lastName as ownerFirstName  
    FROM PersonL HotelGuest  
    INNER JOIN HotelL Hotel ON (HotelGuest.housedAt = Hotel.hotelId)  
    INNER JOIN PersonL HotelOwner ON (Hotel.ownedBy =  
                                         HotelOwner.personId)
```

All underlying queries and communications to the database can be obtained by accessing the main **Webtools** page and selecting **Adjust Debugging Levels**, ticking the **Verbose** checkbox, and clicking **OK**. This change will only last the duration of this session. To make the change permanent, edit the `debug.properties` file in `${OFBizInstallFolder}\framework\base\config` and set `print.verbose=true`.

OFBiz produces the following Query:

```
SELECT HotelGuest.FIRST_NAME, HotelGuest.LAST_NAME, Hotel.HOTEL_NAME,  
      HotelOwner.FIRST_NAME, HotelOwner.LAST_NAME FROM  
      (OFBIZ.PERSON_L HotelGuest INNER JOIN OFBIZ.HOTEL_L Hotel ON  
      HotelGuest.HOUSED_AT = Hotel.HOTEL_ID)  
      INNER JOIN OFBIZ.PERSON_L HotelOwner ON Hotel.OWNED_BY =  
                                         HotelOwner.PERSON_ID
```

The reason for the difference is that OFBiz produces SQL that will work on a number of databases and so the SQL produced is generic. Some databases may store case dependent table names and field names and some might not. For this reason, it was decided that all table names and field names be stored in upper case – removing the case dependency. This is completely removed from us. Unless we are directly querying the database we will never see this, as the framework allows us to use mixed case table and field names and will then convert accordingly, so:

hotel Name becomes HOTEL_NAME

In this case the framework also takes care of the aliasing of field names.

Return to the main **Webtools** page, and select the Entity SQL Processor. This handy tool allows you type SQL in and check the results. This processor "bypasses" the framework changes to the field names and as such the table and field names must be referenced as they are physically stored (example, `HOTEL_NAME`).

The "all including" <alias-all> Element

The `<alias-all>` element includes all fields from the specified member entity. It has the ability to exclude some fields. It can also add a prefix to the included field names to generate meaningful alias names.

The equivalent of the above view entity using <alias-all> elements:

```
<view-entity entity-name="GuestHotelOwnerViewAllL"
            package-name="org.ofbiz.learning">
<member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
<member-entity entity-alias="Hotel" entity-name="HotelL"/>
<member-entity entity-alias="HotelOwner" entity-name="PersonL"/>
<alias-all entity-alias="HotelGuest" prefix="guest">
    <exclude field="id"/><exclude field="housedAt"/>
    <exclude field="someExtraDetails"/>
</alias-all>
<alias-all entity-alias="Hotel" prefix="hotel">
    <exclude field="hotelId"/><exclude field="ownedBy"/>
</alias-all>
<alias-all entity-alias="HotelOwner" prefix="owner">
    <exclude field="personId"/><exclude field="housedAt"/>
    <exclude field="someExtraDetails"/>
</alias-all>
<view-link entity-alias="HotelGuest" rel-entity-alias="Hotel">
    <key-map field-name="housedAt" rel-field-name="hotelId"/>
</view-link>
<view-link entity-alias="Hotel" rel-entity-alias="HotelOwner">
    <key-map field-name="ownedBy" rel-field-name="personId"/>
</view-link>
</view-entity>
```

Don't forget to add the entry into the relevant entitygroup.xml file!

Obviously, it only makes sense to use the <alias-all> element when there are more included fields than excluded ones.

guestPersonId	guestFirstName	guestLastName	hotelHotelName	hotelPostalAddressId	ownerFirstName	ownerLastName
2	Julia	Roberts	Brad's Hotel	9014	Brad	Pitt
1	Brad	Pitt	Hank's Hotel	9300	Tom	Hanks

As can be seen, **Julia Roberts** is housed at **Brad's Hotel**, and **Brad Pitt** himself is housed at **Hank's Hotel**. **Tom Hanks** is not housed anywhere.

The <view-link> Element

View links are the connections between member entities in a view entity. The number of view links in a view entity will always be one less than the number of member entities. If there were 10 people joined together by a single chain, there would be nine chain links.

A view link is defined by a <view-link>. A <view-link> has almost the same syntax and exactly the same structure as a <relation> element. A view link is, after all, really a relation between two member entities in a view entity. Note the similarity between:

```
<view-link entity-alias="HotelGuest" rel-entity-alias="Hotel">
  <key-map field-name="housedAt" rel-field-name="hotelId"/>
</view-link>
```

and

```
<relation type="one" fk-name="PERSON_HOTEL" rel-entity-name="HotelL">
  <key-map field-name="housedAt" rel-field-name="hotelId"/>
</relation>
```

Like the <relation>, each <view-link> element must have at least one <key-map> sub-element.

Inner Joins versus Outer Joins

The <view-link> has an attribute `rel-optional` to specify whether the relation between the joined entities is optional or not. If the relation is optional, then all records in the referencing member entity (specified by the attribute `entity-alias`) will be returned in a query to the view entity. If not, only records for which there are related records in the referenced entity (specified by the attribute `rel-entity-alias`) will be returned. To illustrate, consider these two view entities:

```
<view-entity entity-name="HotelGuests"
            package-name="org.ofbiz.learning">
  <member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
  <member-entity entity-alias="Hotel" entity-name="HotelL"/>
  <alias entity-alias="HotelGuest" name="guestFirstName"
        field="firstName"/>
  <alias entity-alias="HotelGuest" name="guestLastName"
        field="lastName"/>
  <alias entity-alias="Hotel" name="hotelName" field="hotelName"/>
  <view-link entity-alias="HotelGuest" rel-entity-alias="Hotel">
    <key-map field-name="housedAt" rel-field-name="hotelId"/>
  </view-link>
</view-entity>
```

and

```
<view-entity entity-name="PersonHousingLocations"
            package-name="org.ofbiz.learning">
  <member-entity entity-alias="HotelGuest" entity-name="PersonL"/>
  <member-entity entity-alias="Hotel" entity-name="HotelL"/>
  <alias entity-alias="HotelGuest" name="guestFirstName"
```

```
        field="firstName"/>
<alias entity-alias="HotelGuest" name="guestLastName"
      field="lastName"/>
<alias entity-alias="Hotel" name="hotelName" field="hotelName"/>
<view-link entity-alias="HotelGuest" rel-entity-alias="Hotel"
           rel-optional="true">
  <key-map field-name="housedAt" rel-field-name="hotelId"/>
</view-link>
</view-entity>
```

The first view entity will return PersonL records for which field PersonL.housedAt is linked to an existing Hotel record. PersonL records that have field PersonL.housedAt values of null (not residing in hotel) or some non-existent HotelL.hotelId will not be considered at all; the latter case occurs when the view link does not happen to mirror a type one relation. This view entity only considers hotel guests, and not persons who are currently not staying in a hotel.

guestFirstName	guestLastName	hotelName
Brad	Pitt	Hank's Hotel
Julia	Roberts	Brad's Hotel

The second view entity will return every Person record, regardless of whether the person is staying at a hotel or not. For every view entity record returned, a value of null in the field PersonHousingLocations.housedAt will mean that the person is currently not staying in a hotel. This view entity is concerned about the housing or hotel locations of all persons, not just hotel guests.

guestFirstName	guestLastName	hotelName
Brad	Pitt	Hank's Hotel
Julia	Roberts	Brad's Hotel
Tom	Hanks	

Tom Hanks is the only person not housed anywhere.

The first view entity performs an inner join between the entities PersonL and HotelL, or an equi-join specifically (a type of inner join). The second view entity performs a left outer join, a type of outer join where all the records of the left entity (PersonL in this case) are retained and considered.

The <relation> Element

The <relation> elements in a <view-entity> element are exactly like those in an <entity> element.

Applying Functions on Fields

View entity fields can also contain the result of functions, not just member entity field values. The functions available are as follows

- count
- count-distinct
- min
- max
- sum
- avg
- upper
- lower

To use a function, specify it in <alias> element's attribute function. Except for functions upper and lower, all the other functions are aggregate functions that deal with counting.

Counting the Number of Records

The function count counts the number of records. Say we want to count the number of records in the entity `PostalAddress`. In the folder `${component : learning} \ entitydef`, insert into the file `entitymodel.xml` a view entity `TestAggregate`:

```
<view-entity entity-name="TestAggregate"
            package-name="org.ofbiz.learning">
    <member-entity entity-alias="PA" entity-name="PostalAddress"/>
    <alias entity-alias="PA" name="contactMechId" function="count"/>
</view-entity>
```

Assign the view entity to the entity group `org.ofbiz` (edit file `entitygroup.xml`). Restart OFBiz and go to the data maintenance screen for the entity `TestAggregate` to see a single record displayed with a single field containing the total number of records for the entity `PostalAddress`.

Counting Distinct Records

To count the number of distinct records, we use the function `count-distinct`. Let's say we want to count the number of different cities in the `PostalAddress` records. We tell OFBiz to count the number of distinct city values. Go to the folder `${component:learning}\entitydef` and insert a new view entity called `TestCountDistinct` into the file `entitymodel.xml`:

```
<view-entity entity-name="TestCountDistinct"
            package-name="org.ofbiz.learning">
    <member-entity entity-alias="PA" entity-name="PostalAddress"/>
    <alias entity-alias="PA" name="city" function="count-distinct"/>
</view-entity>
```

Assign the view entity to the entity group `org.ofbiz` and restart. Go to the data maintenance screen for the entity `TestCountDistinct` to see a single record displayed with a single field containing the total number of records for the entity `PostalAddress` with distinct city values. Or in short, the number of cities in those records.

Arithmetic Aggregate Functions

Arithmetic aggregate functions operate on numerical values:

- `min`—get the minimum value of all selected records
- `max`—get the maximum value of all selected records
- `sum`—get the sum of the values of all selected records
- `avg`—get the average of the values of all selected records

Uppercase and Lowercase Functions

Functions `upper` and `lower` are string transformation functions, not aggregate functions like the rest. Function `upper` transforms a string into uppercase; `lower` into lowercase. For example:

```
<view-entity entity-name="TestUpperLower"
            package-name="org.ofbiz.learning">
    <member-entity entity-alias="PA" entity-name="PostalAddress"/>
    <alias entity-alias="PA" name="upperToName" field="toName"
          function="upper"/>
    <alias entity-alias="PA" name="lowerAddress1" field="address1"
          function="lower"/>
    <alias-all entity-alias="PA"/>
</view-entity>
```

Grouping for Summary Views

When we want to group a set of data by a field, we use `group-by` in a view entity field.

To group by a field, set the attribute `group-by` of the element `<alias>` to true. Let's say we want to count the number of postal addresses by city:

```
<view-entity entity-name="TestGrouping"
            package-name="org.ofbiz.learning">
<member-entity entity-alias="PA" entity-name="PostalAddress"/>
<alias entity-alias="PA" name="count" field="contactMechId"
      function="count"/>
<alias entity-alias="PA" name="city" group-by="true"/>
</view-entity>
```

Enter the above view entity into the file `entitymodel.xml` in the folder `${component:learning}\entitydef`. Also assign the view entity to the entity group `org.ofbiz`.

Restart OFBiz. Go to the entity data maintenance screen of view entity **TestGrouping** and click **Find All**. Go to the entity data maintenance screen of entity **PostalAddress** to see the postal addresses participating in the view entity, click **Find All** and see all eight cities (plus **A City** for person **OFBiz Researcher**) displayed, with the number of postal addresses for each city.

count	city
4	A City
2	City of Industry
1	Los Angeles
2	New York
1	Open City
4	Orem
1	Roissy
1	Sacramento

Complex Aliases

Complex aliases are fields in view entities that involve an operator that takes in two or more values, so it only makes sense for complex aliases to involve two or more fields. Examples are `+` and `-`, where a typical expression could be `fieldA + fieldB` or `fieldA - fieldB - fieldC`.

Complex aliases are defined by `<alias>` elements that have a sub-element `<complex-alias>`. A `<complex-alias>` element should contain two or more `<complex-alias-field>` sub-elements. A `<complex-alias-field>` points to a member entity's field, where the attribute `entity-alias` specifies the member entity itself and the attribute `field` specifies a field in that member entity.

Nested `<complex-alias>` Elements

A `<complex-alias>` element can also contain another `<complex-alias>` element. You can think of a `<complex-alias>` element as adding parentheses around its fields, so " $(a + b) * ((c - d) / e)$ " would be (note `*` is multiply and `/` is divide):

```
<member-entity entity-alias="EA" entity-name="SomeEntityAlias"/>
<alias name="complexComputedField">
  <complex-alias operator="*">
    <complex-alias operator="+">
      <complex-alias-field entity-alias="EA" field="a"/>
      <complex-alias-field entity-alias="EA" field="b"/>
    </complex-alias>
    <complex-alias operator="/">
      <complex-alias operator="-">
        <complex-alias-field entity-alias="EA" field="c"/>
        <complex-alias-field entity-alias="EA" field="d"/>
      </complex-alias>
      <complex-alias-field entity-alias="EA" field="e"/>
    </complex-alias>
  </complex-alias>
</alias>
```

Operators are Database-Specific

Be careful when using complex aliases because the operator (attribute `operator`) is processed by OFBiz "as is". That is, whatever we enter for `operator` is passed straight to the underlying database in use at the time. Be sure to check that your database supports the operator you are using.

For example, it is not possible to use complex aliases to concatenate strings when using the database MySQL. MySQL does not recognize the `||` operator as a string concatenator. While the following will work with Derby or PostgreSQL, it won't with MySQL:

```
<view-entity entity-name="TestConcat"
            package-name="org.ofbiz.learning">
  <member-entity entity-alias="PA" entity-name="PostalAddress"/>
  <alias-all entity-alias="PA"/>
  <alias name="addressFull">
    <complex-alias operator="||">
      <complex-alias-field entity-alias="PA" field="address1"/>
```

```

<complex-alias-field entity-alias="PA" field="address2"/>
</complex-alias>
</alias>
</view-entity>

```

addressFull	contactMechId	toName	attnName	address1	address2
JAF BuildingPO BOX 1205	NY_DTF_0	NYS Sales Tax Processing		JAF Building	PO BOX 1205

While MySQL does allow for string concatenation with function CONCAT, OFBiz does not support that. The native MySQL syntax is CONCAT(string1, string2), so we might be tempted try to hack it with an operator value of ", ". It won't work. We cannot do this:

```

<view-entity entity-name="TestMySQLConcat"
            package-name="org.ofbiz.learning">
<member-entity entity-alias="PA" entity-name="PostalAddress"/>
<alias-all entity-alias="PA"/>
<alias name="addressFull" function="concat">
    <complex-alias operator=", ">
        <complex-alias-field entity-alias="PA" field="address1"/>
        <complex-alias-field entity-alias="PA" field="address2"/>
    </complex-alias>
</alias>
</view-entity>

```

The attribute function of element `<alias>` cannot be any value other than `min`, `max`, `sum`, `avg`, `count`, `count-distinct`, `upper`, or `lower`.

While OFBiz is largely database-independent, OFBiz was mostly developed and tested with Derby and PostgreSQL. OFBiz is most compatible with PostgreSQL.

Extending Entities

Entities can be extended by using `<extend-entity>` elements. Rather than specifying extensions or enhancements in the original `entitymodel.xml` files (the stock ones that came with OFBiz), we can put the extensions in our own `entitymodel.xml` file (like in the folder `${component:learning}\entitydef`).

Here we take the chance to do some final clean up of the mess we made in Chapter 2, and also experiment with <extend-entity> at the same time. In folder \${component:learning}\entitydef, edit the file entitymodel.xml and insert this:

```
<extend-entity entity-name="PostalAddress">
<field name="planetId" type="id"></field>
<relation type="one" fk-name="POST_ADDR_PLANET"
          rel-entity-name="Planet">
    <key-map field-name="planetId"/>
</relation>
</extend-entity>
```

Go to the folder \${component:party}\entitydef, edit the file entitymodel.xml and delete lines 962 - 963:

```
<field name="planet" type="name"></field>
<field name="planetId" type="id"></field>
```

and then lines 966 - 968:

```
<relation type="one" fk-name="POST_ADDR_PLANET"
          rel-entity-name="Planet">
    <key-map field-name="planetId"/>
</relation>
```

We have now extended the entity `PostalAddress` without modifying the original OFBiz file.

For Java programmers, note that this is not the same as extending or sub-classing in Java programming. The original entity `PostalAddress` is extended; no new entity is created.

Summary

We have now covered all we need to know about creating entities, view entities, and relations between entities or view entities. We have learned much about the Entity Engine in OFBiz. Though this chapter seems a bit heavy, it is worth our time to get to grips with the Entity Engine. Learning the Entity Engine once and for all will allow us to use (or switch to) any of the databases supported by OFBiz.

In this chapter, we looked at:

- Required Administrative Entity Engine concepts like datasources, entity delegators, and entity groups.
- The anatomy of an Entity, that is the `<entity>` element, which can contain `<field>` elements, `<prim-key>` elements, `<relation>` elements, and `<index>` elements.
- Creating our first entity, including the necessary administrative structure in `entitygroup.xml` and `entitymodel.xml` files, plus some additions to our "learning" component's `component.xml` file.
- Using our first entity, creating a user-interface for it, and linking it in a relation to the entity `PostalAddress`.
- Relations between entities via `<relation>` elements, and the corresponding foreign keys
- Indexes on entity fields for fast lookup.
- Relation Types – one, many, and one-nofk
- The anatomy of a view entity, that is the `<view-entity>` element, which can contain `<member-entity>` elements, `<alias>` and `<alias-all>` elements, `<view-link>` elements, and even `<relation>` elements.
- Linking entities (or even other view entities) into a view entity, via the `<member-entity>` elements and the `<view-link>` element.
- Two Types of Linking: Inner Joins and Outer Joins, via the `rel-optional` attribute of `<view-link>` elements.
- Functions (example arithmetic, counting) on view entity fields, via the `function` attribute of the `<alias>` (or more rarely, `<complex-alias-field>`) elements.
- Grouping for summary views, via the `group-by` attribute of the `<alias>` elements.
- Complex aliases (where functions don't cut it), via the `<complex-alias>` and `<complex-alias-field>` elements.
- Extended entities, via `<extend-entity>` elements.

Having learned how to create the Entities and View Entities and the relations between them, we will next look at how to access them and the data in them. We will see how complicated queries can be constructed on the fly by using Dynamic View Entities. We will also learn how improve the application's performance by cutting down database traffic and the number of queries performed by using the Entity Engine cache.

8

Accessing the Entities and View Entities

In the previous chapter, we largely used the "webtools" component in OFBiz to manipulate the data of entities and entity views. In this chapter, we will now look at how to deal with those programmatically. That means we will be firing database queries to the Entity Engine in OFBiz. There are four types of queries: create, update, retrieve, and delete (CRUD).

For the sake of the examples in this chapter we will use the Java language in the BeanShell framework, exploring examples with .bsh files. Although this technique is not necessarily recommended by the OFBiz community, we are able to see what the Java code should look like, without having to stop, recompile, and restart between every example. There are some important differences when accessing the entities in Java versus BeanShell which will be discussed towards the end of the chapter.

Even though Minilang (which will we look at later in the book) does provide many similar functions to those covered in this chapter, it is best that we understand the Java form of these functions first so we will find Minilang even easier later on.

The rate at which the OFBiz framework is progressing is phenomenal and many of the `GenericDelegator` methods that work in OFBiz 4.0 are now deprecated in the "trunk". Although these methods are still fully supported, they may not be the most efficient. For those who have been following these examples throughout the book and are using the latest development trunk as the base code, notes have been added to state which methods should be used in the future.

We will specifically look at:

- How to access entities and View entities (the CRUD)
- Useful utilities for dealing with entities and View entities

- Examining the differences between BeanShell and Java when it comes to entities
- Using Dynamic View entities to build complicated queries programmatically on-the-fly.
- Improve performance by using the Entity Engine cache to dramatically decrease database traffic and the number of queries executed.

Setting-Up Our Playground

In this chapter, we want to focus on learning how to access data using BeanShell scripts. We will set up an environment that easily facilitates our exploration of data access methods in OFBiz, so we won't be distracted by other factors. The environment will consist of:

- a script processor that processes our data access scripts
- a generic display template that displays any data we (our scripts) throw at it

It should be noted that this script processor has no other function other than processing a BeanShell script whose location is passed to it. By using this script processor we are able to cover the examples much faster, as we can bypass the need to create screen widgets and several request / view maps within the controller. Instead, we can write one generic Screen Widget and controller entries to display all of the examples. Do not worry if the code in the script processor is not fully understood, all that is important is the examples that follow the setting up of the processor.

The Script Processor

In the folder \${webapp:learning}\WEB-INF\actions\entityaccess, create a new file processEntityAccessBSF.bsh and enter into it the following:

```
import org.ofbiz.base.location.FlexibleLocation;
import freemarker.ext.beans.BeatnsWrapper;

String scriptName = request.getParameter("scriptName");

if (scriptName == null || scriptName.length() == 0) {
    return "success";
}

String lookInLocation = "component://learning/webapp/learning/
WEB-INF/actions/entityaccess/";

URL location = FlexibleLocation.resolveLocation(lookInLocation +
scriptName);
```

```
if (location == null) {
    context.put("errorMsg", "Script name \'" + scriptName + "\' not
        found at " + lookInLocation);
    return "success";
}

source(location);

bw = BeansWrapper.getDefaultInstance();
bw.setSimpleMapWrapper(true);

Object massaged;
if (data instanceof List) {
    massaged = new ArrayList();
    dataItr = data.iterator();
    while (dataItr.hasNext()) {
        record = (Map) dataItr.next();
        massaged.add(bw.wrap(record));
    }
}
else if (data instanceof Map) {
    massaged = bw.wrap(data);
}

if (massaged != null) {
    context.put("data", massaged);
}

return "success";
```

The Generic Screen

The generic screen that we will use to display the output from our test scripts must now be created.

Creating the Screen Widget

In the folder \${component:learning}\widget\learning, insert into the file LearningScreens.xml a new Screen Widget ProcessEntityAccessBSF:

```
<screen name="ProcessEntityAccessBSF">
    <section>
        <actions>
            <script location="component://learning/webapp/learning/
                WEB-INF/actions/entityaccess/processEntityAccessBSF.bsh"/>
        </actions>
```

```
<widgets>
  <decorator-screen name="main-decorator"
    location="${parameters.mainDecoratorLocation}">
    <decorator-section name="title">
      <label text="Generic Screen for Displaying Data
        Retrieved"/>
    </decorator-section>
    <decorator-section name="body">
      <include-form name="ScriptNameForm"
        location="component://learning/widget/
          learning/LearningForms.xml"/>
      <section>
        <condition><not>
          <if-empty field-name="parameters.scriptName"/>
        </not></condition>
        <widgets>
          <platform-specific><html>
            <html-template location="component://learning/webapp/
              learning/entityaccess/displaydataretrieved.ftl"/>
          </html></platform-specific>
        </widgets>
      </section>
    </decorator-section>
  </decorator-screen>
</widgets>
</section>
</screen>
```

Our BeanShell script `processEntityAccessBSF.bsh` will be operating within a widget BeanShell environment.

Creating the Form Widget

In the folder `${component:learning}\widget\learning`, insert into the file `LearningForms.xml` a new Form Widget `ScriptNameForm`:

```
<form name="ScriptNameForm" type="single"
  target="processEntityAccessBSF">
  <field name="scriptName"><text/></field>
</form>
```

Creating the FreeMarker File

In the folder `${webapp:learning}\entityaccess`, create a new file `displaydataretrieved.ftl` and enter into it the following:

```
<#if errorMsg?has_content>
  <h1 class="errorMessage">${errorMsg}</h1>
</#else>
```

```
<h1>Processed script: "${parameters.scriptName}"</h1>
<%if data?has_content && (data?is_sequence || data?is_hash)>
    <table class="basic-table" style="border: 1px double; margin:
                                              5px">
        <@displayData data=data/>
    </table>
<%else>
    <h1>No data records retrieved.</h1>
<%if>
<%if>

<%macro displayData data>

<%if data?is_sequence>
    <%assign keys = data?first?keys/>
<%else>
    <%assign keys = data?keys/>
<%if>

<%-- Header -->
<tr>
<%list keys as key>
    <td class="dark-grid"><b>${key}</b></td>
<%list>
</tr>

<%-- Data -->
<%if data?is_sequence>
    <%list data as record>
        <tr>
            <%list keys as key>
                <td class="light-grid">${record[key] ! ""}</td>
            <%list>
        </tr>
    <%list>
<%else>
    <tr>
        <%list keys as key>
            <td class="light-grid">${data[key] ! ""}</td>
        <%list>
    </tr>
<%if>

<%macro>
```

Publishing Our Generic Screen

In the folder \${webapp:learning}\WEB-INF, insert into the controller.xml file a new request map:

```
<request-map uri="ProcessEntityAccessBSF">
    <response name="success" type="view"
              value="ProcessEntityAccessBSF"/>
</request-map>
```

and a new view map:

```
<view-map name="ProcessEntityAccessBSF" type="screen"
          page="component://learning/widget/learning/
LearningScreens.xml#ProcessEntityAccessBSF"/>
```

Testing Our Playground

Our playground is set up, and can read any BeanShell scripts we put into the folder \${webapp:learning}\WEB-INF\actions\entityaccess. We shall call this folder \${EntityAccessScriptDir} in this chapter. This folder is where we will be placing all of our scripts that deal with data access. To make sure this is working, let's do a quick test.

In the folder \${EntityAccessScriptDir}, create a new file test.bsh and enter into it this:

```
data = delegator.findAll("PostalAddress");
```

Our script processor looks for the variable data, so our data access scripts must store the results of any data retrieval in that variable.

Fire to webapp learning an http OFBiz request ProcessEntityAccessBSF, and enter test.bsh for the script name. Click **Submit**. We should see all the records for the entity PostalAddress.

Script Name	<input type="text" value="test.bsh"/>					
Submit						
Processed script: "test.bsh"						
contactMechId	toName	attnName	address1	address2	address3	address4
9000	Company XYZ		2003 Open Blvd			
9200	Company XYZ	ZJAA	2003 Open Blvd			
CA_BOE_0	Board of Equalization		P O BOX 942879			
NY_DTF_0	NYS Sales Tax Processing		JAF Building	PO BOX 1205		
9010	Demo Customer Company		2004 Factory Blvd			
9014	Demo Customer		1234 Commerce			

See how easy it is to play with data access on this playground we've set up? We only needed a single file with a single line of code in this case. Let's get on with learning more about data access.

GenericValue Objects

A `GenericValue` object is the equivalent of one record in a database Entity. Every data retrieval will return a `GenericValue` or a List of `GenericValue` objects, if any records are found by the retrieval.

A `GenericValue` object encapsulates three data operations: create, update, and delete. That literally means each database record can be created, updated, or deleted.

While it can easily be understood that we can update or delete a database record (`GenericValue` object) we have in hand, it may not be immediately obvious how to create one. Let us first look at creating a database record.

Creating a Database Record

To create a database record, we need to make a `GenericValue` object. Let's create a new `Planet` record. In the folder `${EntityAccessScriptDir}`, create a new file `createPlanetJupiter.bsh` and enter into it this:

```
import org.ofbiz.entity.*;
import org.ofbiz.base.util.*;

Map planetValues = UtilMisc.toMap("planetId", "JUPITER",
                                  "planetName", "Jupiter");

GenericValue planetGV = delegator.makeValue("Planet", planetValues);
planetGV.create();
data = delegator.findAll("Planet");
```

The `delegator` variable is built into the widget BeanShell (along with some other important variables [<http://docs.ofbiz.org/pages/viewpage.action?pageId=4459>]). We do not need to do anything special like retrieve it from the request like we have to with a Java event (we will learn more about this in the following chapter). It is an object of type `GenericDelegator`, and is the primary data access tool in OFBiz. The `GenericDelegator` is widely used in OFBiz.

In the above code, the method `delegator.makeValue("Planet", planetValues)` creates a `GenericValue` object that represents a data record in memory. The instance method `create()` of the `GenericValue` object (`planetGV`) creates an actual database record, actually persisting the data.

The new `Planet` record should be created successfully now.

 [**Standard SQL Equivalent:**
INSERT INTO Planet (planetId, planetName)
VALUES ("JUPITER", "Jupiter").]

Why a Map is Used?

A map is a collection of key-value pairs, each key being unique in the map. That is exactly how a database record can be represented, the key being the field name and the value being the field value. The variable `planetValues` is a map that represents the values of a single database record for the entity `Planet`.

Rather than use the native `java.util.Map` and `java.util.List` objects, OFBiz makes widespread use of the Javolution high performance implementations `FastMap` and `FastList`. This offers performance advantages, ease of use and can be used in exactly the same way as their native JDK counterparts.

Updating Database Records

We have just created a `Planet` record for `Jupiter`. Imagine that `Jupiter` is now spelled `Joopiter`. We need to update the record with the `planetId` of `JUPITER` accordingly. In the folder `${EntityAccessScriptDir}`, create a new file `updatePlanetJupiter.bsh` and enter into it this:

```
import org.ofbiz.entity.*;
import org.ofbiz.base.util.*;

Map planetValues = UtilMisc.toMap("planetId", "JUPITER");

GenericValue planetGV = delegator.findByPrimaryKey("Planet",
    planetValues);
planetGV.put("planetName", "Joopiter");
planetGV.store();
data = delegator.findAll("Planet");
```

The `GenericDelegator` object `delegator` has a method `findByPrimaryKey` that takes in two parameters:

- a `java.lang.String` (Java class for text strings in general) that is the name of the entity whose records we are looking through
- a `java.util.Map` that is the primary key values that will uniquely identify the one record we want

For the entity `Planet`, the primary key is just the field `Planet.planetId`. Passing in anything other than that will cause the `GenericDelegator` object to throw up an exception (Java term for "error in program execution"). In line 6, we retrieve the same record we created just now.

For those keeping up to date with the latest development trunk, two important lookup methods have changed. The method `findByPrimaryKey` has been replaced with the new `findOne` method and the `findAll` method has been replaced with `findList`.

Standard SQL Equivalent:



```
UPDATE Planet
SET planetName = "Joopiter"
WHERE planetId = "JUPITER"
```

Deleting Database Records

We will now delete the Planet record JUPITER we just created and updated in the previous examples. In the folder \${EntityAccessScriptDir}, create a new file `deletePlanetJupiter.bsh` and enter into it this:

```
import org.ofbiz.entity.*;
import org.ofbiz.base.util.*;

Map planetValues = UtilMisc.toMap("planetId", "JUPITER");

GenericValue planetGV = delegator.findByPrimaryKey("Planet",
                                                 planetValues);

planetGV.remove();
data = delegator.findAll("Planet");
```

The method `remove` on line 7 is an instance method of the class `GenericValue`. And that's all there is to deleting a record.

[ **Standard SQL Equivalent:**
DELETE FROM Planet
WHERE planetId = "JUPITER"]

As long as we have the database record in hand (in the form of a Java object of type `GenericValue`), we can perform update (`store`) or delete (`remove`) operations for that record. The case of creating a database record is a little special. We first need to create a potential `GenericValue` object in memory, and then save the new database record by calling its `create` method. The update and delete operations are executed by calling the methods `store` and `remove`.

We next move on to the fourth type of database operation: retrieval, the "R" in CRUD database operations.

Retrieving Database Records

So far, we have only seen two styles of retrievals: find one (unique record) and find all. Finding a single unique database record requires that we use a primary key value to lookup that record. The primary key values of all records of an Entity are unique; this constraint is enforced by the underlying RDBMS. Finding all records is simple and obvious. Let us look at other styles of retrieving records.

In the following sections, we will be assuming that there are three `Planet` entity records in the database: MARS, EARTH, and PLUTO. If there aren't, you can use the script `createPlanetJupiter.bsh` and change the `planetId` and `planetName` fields in the script to create them. Create more `Planet` records if you like.

Find Records by Conditions

There really are only two styles of retrieval: unconditional and conditional. The `GenericDelegator`'s instance method `findAll` is an unconditional retrieval. The instance method `findByPrimaryKey` has a single condition that is to find the one record that has a particular primary key value.

We use the method `findByCondition` in the `GenericDelegator` class. Let's create our first example. In the folder `${EntityAccessScriptDir}`, create a new file `findByCondition.bsh` and enter into it this:

```
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

condition = new EntityExpr("planetId", EntityOperator.EQUALS,
                           "EARTH");
data = delegator.findByCondition("Planet", condition, null, null);
```

Fire to webapp learning an http OFBiz request `ProcessEntityAccessBSF`, and enter the script name `findByCondition.bsh`. Submit. We should see a single `Planet` record that has the value `EARTH` in the field `planetID`.

planetId	planetName	fromDate	thruDate	la
EARTH	Earth			2

[ Standard SQL Equivalent:
 SELECT * FROM Planet
 WHERE planetId = "EARTH"]

The instance method `findByCondition` takes four parameters:

- `entityName` of type `java.lang.String`, which is the name of the entity whose records we want to retrieve.
- `entityCondition` of type `org.ofbiz.entity.condition.EntityCondition`, which is the condition to constrain the retrieval.
- `fieldsToSelect` of type `java.util.Collection` (the more general form of `java.util.List`, its superclass), which is the fields of the records to retrieve (null for all).
- `orderBy` of type `java.util.List`, which is the list of fields by which we want the retrieved records to be ordered (null for no ordering).

In the trunk, `findByCondition` and its variants have been replaced with `findList`.

The `entityCondition` parameter can be complex. We will look at that next.

Conditions

Conditions in OFBiz are represented by objects of type `org.ofbiz.entity.condition.EntityCondition`. `EntityCondition` is an abstract class, which means it is not concrete and cannot be used directly. It contains abstract methods that are like mere blueprints, which need to be concretely implemented by a subclass.

There are a few forms of conditions in OFBiz. One form is the raw SQL string, which is in the native database access language we are insulated from by OFBiz, represented by an object of type `org.ofbiz.entity.condition.EntityWhereString`. Another form is an object of type `org.ofbiz.entity.condition.EntityExpr`. Both classes `EntityWhereString` and `EntityExpr` extend and implement the abstract (blueprint) class `EntityCondition`.

A condition expression has the form `someField = someValue`, where the operator "`=`" can be replaced with others like "`>`" (greater than) and "`<`" (lesser than). A condition expression is made up of a left hand-side operand, an operator, and a right-hand-side operand, in that order. The simplest way to create an `EntityExpr` object is to instantiate (create an instance) one using these three parameters:

- `lhs` of type `java.lang.String`, which is the left hand-side operand.
- `operator` of type `org.ofbiz.entity.condition.EntityComparisonOperator`, which is the comparison operator between the two operands.
- `rhs` of type `java.lang.Object`.

An example we have just seen is this:

```
new EntityExpr("planetId", EntityOperator.EQUALS, "EARTH");
```

In the development trunk, `EntityExpr` has recently been replaced with `EntityCondition.makeCondition`.

Comparison Operators (`EntityComparisonOperator`)

The operator in a condition is represented by an `EntityComparisonOperator` object. We should not create `EntityComparisonOperator` objects ourselves, but use the pre-defined ones in class `org.ofbiz.entity.condition.EntityOperator`. To access one such object, we do something like `EntityOperator.EQUALS`, where `EQUALS` is a field in the class `EntityOperator`.

Java classes can have fields (not database record fields, in this case), besides having methods. Fields and methods are members of a class; fields are like attributes of a class, while methods are the behavior. The ". " operator accesses any publicly accessible members in the class. Using the operator on an object will access instance members, while using it on a class will access class members.

The list of operators are:

- EQUALS
- NOT_EQUAL
- LESS_THAN
- GREATER_THAN
- LESS_THAN_EQUAL_TO (less than or equal to)
- GREATER_THAN_EQUAL_TO
- IN
- NOT_IN
- BETWEEN
- LIKE
- NOT_LIKE
- NOT

We will ignore the NOT operator. It is never used in OFBiz, nor does it have any equivalent in SQL. The original author of this probably intended it to be a logical "not" (unary operator), which will mean it should be of type EntityJoinOperator (explained later). Still, unary operators (where left hand-side operand is omitted) are not yet supported in OFBiz, and still need some work.

We will be exploring the use of the above operators next. The operators can be grouped by the kinds of right-hand-side values or operands they operate on.

Single-Value Operators

Operators that operate on single right hand-side values are EQUALS, NOT_EQUAL, LESS_THAN, GREATER_THAN, LESS_THAN_EQUAL_TO, and GREATER_THAN_EQUAL_TO. For example, create in folder \${EntityAccessScriptDir} a new file singeValueOperator.bsh and enter into it this:

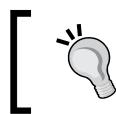
```
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

condition = new EntityExpr("planetId", EntityOperator.NOT_EQUAL,
                           "EARTH");
data = delegator.findByCondition("Planet", condition, null, null);
```

Accessing the Entities and View Entities

The above will pull down all `Planet` records whose field `planetId` is not equal to EARTH.

planetId	planetName	fromDate	thruDate	lastUpdatedStamp	lastUpdatedTxStamp	createdStamp	createdTxStamp
MARS	Mars			2008-06-17 18:06:53.859	2008-06-17 18:06:53.859	2008-06-17 18:06:53.859	2008-06-17 18:06:53.859
PLUTO	Pluto			2008-06-17 18:07:05.046	2008-06-17 18:07:05.046	2008-06-17 18:07:05.046	2008-06-17 18:07:05.046



Standard SQL Equivalent:

```
SELECT * FROM Planet
WHERE planetId != "EARTH"
```

Set-Value Operators

Operators that operate on set right hand-side values are `IN` and `NOT_IN`. Records that match any values in the set will be retrieved. For an example, create in folder `$(EntityAccessScriptDir)` a new file `setValueOperator.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

condition = new EntityExpr("planetId", EntityOperator.IN,
                           UtilMisc.toList("EARTH", "PLUTO"));
data = delegator.findByCondition ("Planet", condition, null, null);
```

The above will pull down all `Planet` records whose field `planetId` is equal to any of the values in the list. The list contains EARTH and PLUTO.

planetId	planetName	fromDate	thruDate	lastUpdatedStamp	lastUpdatedTxStamp	createdStamp	createdTxStamp
EARTH	Earth			2008-06-17 18:06:34.968	2008-06-17 18:06:34.968	2008-06-17 18:06:34.968	2008-06-17 18:06:34.968
PLUTO	Pluto			2008-06-17 18:07:05.046	2008-06-17 18:07:05.046	2008-06-17 18:07:05.046	2008-06-17 18:07:05.046



Standard SQL Equivalent:

```
SELECT * FROM Planet
WHERE planetId IN ("EARTH", "PLUTO")
```

`UtilMisc.toList` can take up to six parameters to return a `java.util.List` object composed of those parameter values.

There is a difference between **Sets** and **Lists**. Sets cannot have duplicate values, but lists can. Although it makes no sense in this case to pass in duplicate values to the method `findByCondition`, real-world usage patterns sometimes do. Imagine an end-user keying in a set of values with which to find records. It isn't always realistic to expect the end-user to ensure zero duplicates in that set? Hence, to cater for more general use, the method `findByCondition` for set-value operators accepts `java.util.Collection` type parameters, rather than just `java.util.Set`.

Passing in a set of unique values is effectively the same as passing in a list of possibly duplicate values in this case. In OFBiz 4.0, `UtilMisc` does not yet provide a utility for creating sets—although this functionality will be present in OFBiz 5.0.

Text-Value Operators

The operator that operates on text right hand-side values is `LIKE` and `NOT_LIKE`. Records with text fields whose values contain a substring matching the text value, which is a pattern, will be retrieved. For an example, create a new file `textValueOperator.bsh` in the folder `${EntityAccessScriptDir}` and enter into it this:

```
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

condition = new EntityExpr("planetName", EntityOperator.LIKE,
                           "%AR%") ;
data = delegator.findByCondition("Planet", condition, null, null);
```

The right-hand-side value is a pattern made up of text, plus possibly two kinds of special characters called **wildcard characters**. The "%" character represents "any number of characters". The "_" character represents "any one character".

In the context of our three `Planet` records, the pattern "%ar%" will match `Mars` and `Earth`. The pattern "%ar_" will only match `Mars`.



Standard SQL Equivalent:

```
SELECT * FROM Planet
WHERE planetName LIKE "%ar%"
```

Condition Lists

Condition lists are another form of Entity condition (`EntityCondition`) in OFBiz, the other two being `EntityWhereString` and `EntityExpr` which were discussed in previous sections. It is represented by an object of type `org.ofbiz.entity.condition.EntityConditionList`.

A condition list is a combined list of Entity conditions joined together by a logical operator AND or OR. In plain English, it's like saying "find me records that satisfy this condition or that condition". An `EntityConditionList` object is created with two parameters:

- `conditionList` of type `java.util.List`, which should be a list of `EntityCondition` objects.
- `operator` of type `EntityJoinOperator`, which is the joiner.

Condition Joiners (`EntityJoinOperator`)

Condition joiners are operators that join two or more conditions together. Technically speaking, these operators are really "logical operators". Condition joiners are represented by objects of type `org.ofbiz.entity.condition.EntityJoinOperator`. Like `EntityComparisonOperator`, we should not create `EntityJoinOperator` objects ourselves, but use the pre-defined ones in `EntityOperator`.

There are only two joiners in OFBiz: AND and OR.

The AND Joiner

In the folder `${EntityAccessScriptDir}`, create a new file `andJoiner.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%ar%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%t%"));
conditions = new EntityConditionList(conditionList,
                                     EntityOperator.AND);
data = delegator.findByCondition("Planet", conditions, null, null);
```

In the context of our three `Planet` records, the above should only return planet Earth.

[ **Standard SQL Equivalent:**
SELECT * FROM Planet
WHERE planetName LIKE "%ar%" AND planetId LIKE "%t%"]

The OR Joiner

In the folder \${EntityAccessScriptDir}, create a new file `orJoiner.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%s%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%P%"));
conditions = new EntityConditionList(conditionList,
                                      EntityOperator.OR);
data = delegator.findByCondition("Planet", conditions, null, null);
```

In the context of our three `Planet` records, the above should return `Mars` and `Pluto`.

 **Standard SQL Equivalent:**

```
SELECT * FROM Planet
WHERE planetName LIKE "%ar%" OR planetId LIKE "%t%"
```

Nested Condition Lists

Recall that the instantiation method for class `EntityConditionList` takes a list of `EntityCondition` objects. Since `EntityConditionList` objects are a form of `EntityCondition` objects, we can join them together to form complex conditions.

In the folder \${EntityAccessScriptDir}, create a new file `nestedJoiner.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.*;
import org.ofbiz.entity.condition.*;

conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%ar%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%m%"));
condition1 = new EntityConditionList(conditionList,
                                      EntityOperator.OR);

condition2 = new EntityConditionList
    (UtilMisc.toList(
        new EntityExpr("planetName", EntityOperator.LIKE, "%s%"),
        new EntityExpr("planetName", EntityOperator.LIKE, "%P%"),
        EntityOperator.OR));
combined = new EntityConditionList(UtilMisc.toList(condition1,
                                                 condition2), EntityOperator.AND);

data = delegator.findByCondition("Planet", combined, null, null);
```

The first condition list `condition1` covers Earth and Mars. The second list `condition2` covers Mars and Pluto. Since the two lists are combined by the `AND` joiner, only Mars is retrieved.

 **Standard SQL Equivalent:**

```
SELECT * FROM Planet
WHERE (planetName LIKE "%ar%" OR planetName LIKE "%t%")
AND (planetName LIKE "%s%" OR planetName LIKE "%P%")
```

Selecting Fields to be Retrieved

So far, we have left empty the third parameter of `GenericDelegator`'s instance method `findByCondition`. That tells the `findByCondition` method to retrieve every field present in the Entity being searched. To limit the number of fields retrieved, we pass in a `List` of specific field names. In the latest version, the `fieldsToSelect` has been changed to `java.util.Set`.

In the folder `${EntityAccessScriptDir}`, create a new file `selectFields.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
fields = UtilMisc.toList("planetId", "planetName");
data = delegator.findByCondition("Planet", null, fields, null);
```

The above will pull down all `Planet` records, but show only two fields for each one: `planetId` and `planetName`.

Note how passing in a null condition (second parameter) to the method `findByCondition` means to "find all records".

 **Standard SQL Equivalent:**

```
SELECT planetId, planetName FROM Planet
```

Ordering the Retrieved Records

The fourth parameter of `GenericDelegator`'s instance method `findByCondition` takes in a list of field names, and orders the retrieved records by those fields. If there is more than one field in the list, ordering is performed by the first field, then the second, and so on.

In the folder `${EntityAccessScriptDir}`, create a new file `orderedRecords.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;

fields = UtilMisc.toList("planetId DESC");
data = delegator.findByCondition("Planet", null, null, fields);
```

The `DESC` keyword means to arrange records in descending order. If the `DESC` keyword is omitted, the default behavior is to arrange in ascending order.

You may, from time to time, see:

```
fields = UtilMisc.toList("-planetId");
```

The negative sign has exactly the same effect as the `DESC`.

The above will pull all `Planet` records sorted by field `planetId` in descending order.

planetId	planetName	fromDate	thruDate	lastUpdatedStamp	lastUpdatedTxStamp	createdStamp	createdTxStamp
PLUTO	Pluto			2008-06-17 18:07:05.046	2008-06-17 18:07:05.046	2008-06-17 18:07:05.046	2008-06-17 18:07:05.046
MARS	Mars			2008-06-17 18:06:53.859	2008-06-17 18:06:53.859	2008-06-17 18:06:53.859	2008-06-17 18:06:53.859
EARTH	Earth			2008-06-17 18:06:34.968	2008-06-17 18:06:34.968	2008-06-17 18:06:34.968	2008-06-17 18:06:34.968



Standard SQL Equivalent:

```
SELECT * FROM Planet ORDER BY planetId DESC
```

Dealing with Grouped and Summary Views

The instance method `findByCondition` also has a more complicated form that allows us to do grouped and summary views. The query section we're dealing with here is called the `HAVING` clause. So far, we have only been dealing with the `WHERE` clause, that is "where these conditions are met". The `HAVING` clause is like the `WHERE` clause, except it is performed on grouped retrieval results, or more specifically on aggregate fields. Some hands-on examples will shed more light on this.

In the folder `${EntityAccessScriptDir}`, create a new file `havingCondition.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.condition.*;

fields = UtilMisc.toList("count", "city");
orderBy = UtilMisc.toList("-count");
data = delegator.findByCondition("TestGrouping", null, fields,
                                orderBy);
```

Notice the use of the `TestGrouping` Entity we created in the previous chapter.

That will pull down all `PostalAddress` records grouped by the field `PostalAddress.city`. That means we see a summary view of all the records summarized into "the number of postal addresses for each city that exists in the records".

count	city
4	Orem
4	A City
2	New York
2	City of Industry
1	Sacramento
1	Roissy
1	Open City
1	Los Angeles

 **Standard SQL Equivalent:**

```
SELECT COUNT(PA.contactMechId), PA.city
FROM PostalAddress PA
GROUP BY PA.city
ORDER BY COUNT(PA.contactMechId) DESC
```

Say we now want only those cities for which there is more than one postal address. We might try something like introducing an Entity condition in the `WHERE` clause by replacing lines 6-7 with this:

```
condition = new EntityExpr("count", EntityOperator.GREATER_THAN, 1);
data = delegator.findByCondition("TestGrouping", condition, fields,
                                orderBy);
```

But we get an error in `ofbiz.log` saying: **SQL Exception... (Invalid use of an aggregate function.)**. As mentioned, only the `HAVING` clause is meant to deal with aggregate fields, not the `WHERE` clause. The field `TestGrouping.count` is an aggregate field. Undo the above change.

Another form of the instance method `findByCondition`, the correct form to use here, takes six parameters:

- `entityName` of type `java.lang.String`, which is the name of the Entity whose records we want to retrieve.
- `whereEntityCondition` of type `org.ofbiz.entity.condition.EntityCondition`, which is the condition to put in the `WHERE` clause.
- `havingEntityCondition` of type `org.ofbiz.entity.condition.EntityCondition`, which is the condition to put in the `HAVING` clause.
- `fieldsToSelect` of type `java.util.Collection`, which is the fields of the records to retrieve (null for all).
- `orderBy` of type `java.util.List`, which is the list of fields by which we want the retrieved records to be ordered (null for no ordering).
- `findOptions` of type `org.ofbiz.entity.util.EntityFindOptions`, which we will not cover yet.

Now, replace lines 6-7 with:

```
condition = new EntityExpr("count", EntityOperator.GREATER_THAN, 1);
data = delegator.findByCondition("TestGrouping", null, condition,
                                fields, orderBy, null);
```

count	city
4	Orem
4	A City
2	New York
2	City of Industry

Standard SQL Equivalent:

```
SELECT COUNT(PA.contactMechId), PA.city
FROM PostalAddress PA
GROUP BY PA.city
HAVING COUNT(PA.contactMechId) > 1
ORDER BY COUNT(PA. contactMechId) DESC
```



Tools for Common Styles of Conditions

In OFBiz, there are several tools that allow us to easily and quickly put together common styles of conditions.

Conditions Joined by AND

The instance method `findByAnd` takes in a `List` of Entity conditions and combines them with the operator `EntityOperator.AND`. It then performs a retrieval query with the combined conditions. The instance method `findByAnd` in `GenericDelegator` automatically constructs an `EntityConditionList` object from a `java.util.List` object, and then calls the method `findByCondition` internally. In the folder `${EntityAccessScriptDir}`, create a new file `findByAnd.bsh` and enter into it this:

```
import org.ofbiz.entity.condition.*;
import org.ofbiz.base.util.*;

conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%a%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%r%"));

data = delegator.findByAnd("Planet", conditionList);
```

Lines 4-8 in the above are the same as:

```
conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%a%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%r%"));

conditions = new EntityConditionList(conditionList,
                                      EntityOperator.AND);

data = delegator.findByCondition("Planet", conditions, null, null);
```

The `findByAnd` method retrieves all fields; there is no way to select or omit fields.

Special Usage with `java.util.Map`

The second parameter can also accept a map of key-value pairs, instead of a list of Entity conditions. Each key-value pair is a single entity condition with the `EntityOperator.EQUALS` operator.

In the folder `${EntityAccessScriptDir}`, create a new file `findByAndMap.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;

conditions = UtilMisc.toMap("planetId", "EARTH", "planetName",
                           "Earth");

data = delegator.findByAnd("Planet", conditions);
```

Since we are dealing with maps, it is not possible to have multiple conditions for the same field. Maps cannot have duplicate keys. Updating a map using `map.put("someFieldName", "somenewValue")` will update the key-pair that has a key of `someFieldName` so that its value is now `somenewValue`. Any old values are overwritten.

Ordering Retrieved Records

In this situation, when using a map to perform the lookup, the `findByAnd` method can take in an optional third parameter that is a list of fields by which to order the retrieved records. Edit the file `findByAnd.bsh` and replace the final line with:

```
orderBy = UtilMisc.toList("-planetName");
data = delegator.findByAnd("Planet", conditions, orderBy);
```

Conditions Joined by OR

The instance method `findByOr` takes in a list of Entity conditions and combines them with the operator `EntityOperator.OR`. It then performs a retrieval query with the combined conditions. Internally, the `findByOr` method works the same as the `findByAnd` method. In the folder `${EntityAccessScriptDir}`, create a new file `findByOr.bsh` and enter into it this:

```
import org.ofbiz.entity.condition.*;
import org.ofbiz.base.util.*;

conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%s%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%P%"));

data = delegator.findByOr("Planet", conditionList);
```

Lines 4-8 above are the same as:

```
conditionList = UtilMisc.toList
    (new EntityExpr("planetName", EntityOperator.LIKE, "%s%"),
     new EntityExpr("planetName", EntityOperator.LIKE, "%P%"));
```

```
conditions = new EntityConditionList(conditionList,
                                     EntityOperator.OR);

data = delegator.findByCondition("Planet", conditions, null, null);
```

The `findByOr` method retrieves all fields; there is no way to select or omit fields. This method also accepts an optional third parameter for records ordering just like the method `findByAnd` does. Also, this method allows the special usage style with maps, just like `findByAnd`.

Counting the Number of Records Retrieved

Sometimes, we just want a quick count of the number of records that match our `find by conditions`. A typical usage is like this. Say we have a retrieval query that will potentially return thousands of records. We want to display only 10 records at any one time, so the end-user can scroll through the thousands of records one (manageable) handful at a time. We also want to show a number indicating exactly how many records there are in total. This will be a good case in which to use a quick count.

In the folder `${EntityAccessScriptDir}`, create a new file `findCount.bsh` and enter into it this:

```
import org.ofbiz.entity.condition.*;
import org.ofbiz.base.util.*;

conditions = new EntityConditionList
(UtilMisc.toList
(new EntityExpr("planetName", EntityOperator.LIKE, "%a%"),
 new EntityExpr("planetName", EntityOperator.LIKE, "%r%")),
EntityOperator.OR);

count = delegator.findCountByCondition("Planet", conditions, null);
data = UtilMisc.toMap("count", count);
```

The above will retrieve two `Planet` records `Earth` and `Mars`, so the result will be a single field `count` of value 2.

Note how the code above doesn't require the use of a View Entity, like we used with `TestGrouping` earlier in the chapter. We do not need to create a View Entity for every query we want to count.

The second parameter of the instance method `findCountByCondition` takes in Entity conditions for the `WHERE` clause and the third parameter—the `HAVING` clause. We look at the `HAVING` clause for this instance method next.

In the folder `${EntityAccessScriptDir}`, create a new file `findCountHaving.bsh` and enter into it this:

```
import org.ofbiz.entity.condition.*;
import org.ofbiz.base.util.*;

condition = new EntityExpr("count", EntityOperator.GREATER_THAN, 1);
count = delegator.findCountByCondition("TestGrouping", null,
                                       condition);
data = UtilMisc.toMap("count", count);
```

The above code will count the number of cities that have more than one postal address.

We are using the View Entity `TestGrouping` here, because there is no way in OFBiz to do a `GROUP BY` database query outside of a View Entity.

The instance method `findCountByAnd` works in a similar way to the form of `findByAnd` that takes in a `java.util.Map` for the second parameter.

In the development trunk, `findCountByAnd` and its variants have been deprecated and replaced with `findCountByCondition`.

OFBiz Date Condition

OFBiz employs a uniform way to deal with database records that may expire over time. This mechanism involves two dates: a `fromDate` and a `thruDate`. This mechanism can also serve as a convenient means to keep an audit trail for records in OFBiz.

Finding Records Active at this Moment

One common style of conditional queries we need is to retrieve records that have not expired at this moment. In the folder `${EntityAccessScriptDir}`, create a new file `filterByDate.bsh` and enter into it this:

```
import org.ofbiz.entity.util.*;
conditions = EntityUtil.getFilterByDateExpr();
data = delegator.findByCondition("Planet", conditions, null, null);
```

Expire and unexpire the `Planet` records to experiment with the above.

Finding Records Active at Some Particular Moment

The class method `getFilterByDateExpr` of class `EntityUtil` can take in an optional parameter `moment` of type `java.util.Date`. That allows us to find active records at any particular moment.

In the folder `${EntityAccessScriptDir}`, create a new file `filterBySomeDate.bsh` and enter into it this:

```
import org.ofbiz.entity.util.*;
import org.ofbiz.base.util.*;

date = UtilDateTime.toDate("12/30/2008 00:00:00");
conditions = EntityUtil.getFilterByDateExpr(date);
data = delegator.findByCondition("Planet", conditions, null, null);
```

Experiment with different dates for the variable `date`. We now need to understand how to use `UtilDateTime` to construct `java.util.Date` objects.

UtilDateTime Class Method "toDate"

The class method `toDate` has four forms. The first form takes in a single string parameter with the format `MM/DD/YYYY HH:mm:ss` where:

- `MM` is a two-digit month
- `DD` is a two-digit day of the month
- `YYYY` is a four-digit year
- `HH` is a two-digit hour (24-hour format)
- `mm` is a two-digit minute
- `ss` is a two-digit second

The second form takes in two string parameters, the first being a date string of format `MM/DD/YYYY` and the second a time string of format `HH:mm:ss`.

The third form takes in six string parameters, in the same order as the above formats: month, day, year, hour, minute, and second.

The fourth form is like the third, except it takes in six `int` parameters instead. Type `int` means integer.

As an alternative you could use `UtilDateTime.toTimestamp("MM/DD/YYYY HH:mm:ss")` to obtain a `java.sql.Timestamp`. The method `EntityUtil.getFilterByDateExpr(date)` has been overloaded to accept either.

Dealing with Field Names Other Than "fromDate" and "thruDate"

We can explicitly specify alternative or custom field names for `fromDate` and `thruDate`. We can do:

```
conditions = EntityUtil.getFilterByDateExpr("myFromDate",
                                             "myThruDate");
```

and

```
conditions = EntityUtil.getFilterByDateExpr(date, "myFromDate",
                                             "myThruDate");
```

We rarely find the need to create entities that have non-conventional from/thru date field names (not `fromDate` and `thruDate`). However, when constructing View Entities, we do often find the need to create alias names for those date fields. Member entities in a View Entity may often have their own `fromDate` and `thruDate` fields, which will need to be renamed to coexist in a View Entity.

Getting Related Records

We often need to get records related to a particular record. For example, we may want to find out the planet name of the planet related to a `PostalAddress` record. Or we may want to find all `PostalAddress` records related to a particular `Planet` record.

For this section, we will need to assign some postal addresses to some planets. Go to the **Entity Data Maintenance** screen for entity `PostalAddress`. Edit some records and set their fields `planetId` to point to any of the three `Planet` records MARS, EARTH, and PLUTO.

One-to-One Relations

The entity `PostalAddress` has a one-to-one relation with the entity `Planet`. Once we get a hold of a `PostalAddress` record in the form of a `GenericValue` object, we call the object's `getRelatedOne` method (an instance method of `GenericValue`) to get the one related `Planet` record.

In the folder `${EntityAccessScriptDir}`, create a new file `getRelatedOne.bsh` and enter this:

```
import org.ofbiz.entity.*;
import org.ofbiz.base.util.*;

data = new ArrayList();

String[] fields = new String[]
 {"contactMechId", "address1", "address2"};

postalAddresses = delegator.findAll("PostalAddress");
paIter = postalAddresses.iterator();
while (paIter.hasNext()) {
 GenericValue pa = paIter.next();
```

```
Map dataRecord = UtilMisc.toMap  
    (fields[0], pa.get(fields[0]),  
     fields[1], pa.get(fields[1]),  
     fields[2], pa.get(fields[2]));  
  
GenericValue planet = pa.getRelatedOne("Planet");  
String planetName = null;  
if (planet != null)  
    planetName = planet.get("planetName");  
  
dataRecord.put("planetName", planetName);  
  
data.add(dataRecord);  
}
```

On line 17, the instance method `getRelatedOne` of a `GenericValue` object is called to get the related `Planet` record in the one-to-one relation. If no record is found, that instance method returns null. That is why we need to test for null on line 19.

contactMechId	address1	address2	planetName
9000	2003 Open Blvd		Pluto
9200	2003 Open Blvd		Pluto
CA_BOE_0	P O BOX 942879		Mars
NY_DTF_0	JAF Building	PO BOX 1205	Mars
9010	2004 Factory Blvd		Mars
9014	1234 Commerce Way		Earth
9011	2004 Factory Blvd		

Our playground looks at the variable `data` and displays it, if it's of type `java.util.List` or `java.util.Map`, the former taken to represent a list of records and the latter a single record. The class `java.util.ArrayList` is an implementation of the interface `List`.

One-to-Many Relations

The Entity `Planet` has a one-to-many relation with the Entity `PostalAddress`. Once we get a hold of a `Planet` record in the form of a `GenericValue` object, we call the object's `getRelated` instance method to get the list of related `PostalAddress` records.

In the folder `${EntityAccessScriptDir}`, create a new file `getRelated.bsh` and enter into it this:

```

import org.ofbiz.entity.*;
import org.ofbiz.base.util.*;

data = new ArrayList();

String[] fields = new String[]
    {"contactMechId", "address1", "address2"};

planets = delegator.findAll("Planet");
plIter = planets.iterator();
while (plIter.hasNext()) {
    GenericValue pl = plIter.next();

    List postalAddresses = pl.getRelated("PostalAddress");
    paIter = postalAddresses.iterator();
    while (paIter.hasNext()) {
        GenericValue pa = paIter.next();
        Map dataRecord = UtilMisc.toMap
            ("planetId", pl.get("planetId"),
             "planetName", pl.get("planetName"),
             fields[0], pa.get(fields[0]),
             fields[1], pa.get(fields[1]),
             fields[2], pa.get(fields[2]));
        data.add(dataRecord);
    }
}
}

```

Script Name	<input type="text" value="getRelated.bsh"/>			
<input type="button" value="Submit"/>				
Processed script: "getRelated.bsh"				
planetId	planetName	contactMechId	address1	address2
EARTH	Earth	9014	1234 Commerce Way	
MARS	Mars	CA_BOE_0	P O BOX 942879	
MARS	Mars	NY_DTF_0	JAF Building	PO BOX 1205
PLUTO	Pluto	9000	2003 Open Blvd	
PLUTO	Pluto	9200	2003 Open Blvd	

Utilities for Post-Query processing

The class `org.ofbiz.entity.util.EntityUtil` contains several post-query operations. We will discuss a few here. These post-query operations let us avoid requerying the database when we only need to tweak the query results a little.

Post-Query Ordering

At times, we may need to display various types of ordering for a given set of query results. We could fire several queries to the database with different "order by" parameters but that would take time. In general, a database query accesses the hard disk, and is significantly slower than an operation performed in memory. When a database query is answered, the retrieved records are usually stored in the computer's memory.

So there are two ways of ordering a set of database records: through a re-query to the database, or through EntityUtil's class method `orderBy`. For each of the two ways, let us count the number of operations we can execute within two seconds. In folder `EntityAccessScriptDir`, create a new file `orderByTiming.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.condition.*;
import org.ofbiz.entity.util.*;

descOrder = UtilMisc.toList("city DESC");
ascOrder = UtilMisc.toList("city ASC");

long testTime = 2 * 1000; // 2 seconds
long endTime = System.currentTimeMillis() + testTime;
long count = 0;
while (System.currentTimeMillis() <= endTime) {
    data = delegator.findByCondition
        ("PostalAddress", null, null, descOrder);
    data = delegator.findByCondition
        ("PostalAddress", null, null, ascOrder);
    count++;
}
Debug.logInfo("Timing orderBy.bsh with database access for " +
              testTime + " ms: " + count, "");
vendTime = System.currentTimeMillis() + 2000;
count = 0;
while (System.currentTimeMillis() <= vendTime) {
    EntityUtil.orderBy(data, descOrder);
    EntityUtil.orderBy(data, ascOrder);
    count++;
}
Debug.logInfo("Timing orderBy.bsh with EntityUtil for " + testTime +
              " ms: " + count, "");
```

In the folder `${OFBizInstallFolder}\runtime\logs`, look into the file `ofbiz.log` and search for the string **Timing orderBy**. We will probably see something like this:

```
Timing orderBy with database access for 2000 ms: 564
Timing orderBy with EntityUtil for 2000 ms: 5711
```

So in two seconds the `orderBy` with database access was performed **564** times whereas the `orderBy` using the post-query processing was performed **5711** times. Clearly, doing a re-query to the database is much slower (about 10 times) than re-ordering a set of retrieved records in memory.

The class method `orderBy` of the class `EntityUtil` takes the following parameters:

- values of type `java.util.Collection`, which is the list or set of `GenericValue` objects we want to sort.
- `orderBy` of type `java.util.List`, which is the list of fields by which we want values to be ordered.

Post-Query Filtering by Conditions

The class method `filterByCondition` of the class `EntityUtil` is analogous to the instance method `findByCondition` of the class `GenericDelegator`, except it operates on a list of retrieved database records in memory rather than doing a re-query to the database.

In the folder `${EntityAccessScriptDir}`, create a new file `filterByConditionTiming.bsh` and enter into it this:

```
import org.ofbiz.base.util.*;
import org.ofbiz.entity.condition.*;
import org.ofbiz.entity.util.*;

conditionList1 = UtilMisc.toList
    (new EntityExpr("city", EntityOperator.LIKE, "O%"),
     new EntityExpr("city", EntityOperator.LIKE, "%o"));
conditionList2 = UtilMisc.toList
    (new EntityExpr("city", EntityOperator.LIKE, "C%"),
     new EntityExpr("city", EntityOperator.LIKE, "%y"));
condition1 = new EntityConditionList(conditionList1,
                                     EntityOperator.AND);
condition2 = new EntityConditionList(conditionList2,
                                     EntityOperator.AND);

orderBy = UtilMisc.toList("city");

long testTime = 2 * 1000; // 2 seconds
long endTime = System.currentTimeMillis() + testTime;
long count = 0;
```

```
        while (System.currentTimeMillis() <= endTime) {
            data = delegator.findByCondition
                ("PostalAddress", condition1, null, orderBy);
            data = delegator.findByCondition
                ("PostalAddress", condition2, null, orderBy);
            count++;
        }
        Debug.logInfo("Timing filterByCondition with database access for " +
                      testTime + " ms: " + count, "");
        allRecords = delegator.findAll("PostalAddress");
        endTime = System.currentTimeMillis() + 2000;
        count = 0;
        while (System.currentTimeMillis() <= endTime) {
            data = EntityUtil.orderBy
                (EntityUtil.filterByCondition(allRecords, condition1),
                 orderBy);
            data = EntityUtil.orderBy
                (EntityUtil.filterByCondition(allRecords, condition2),
                 orderBy);
            count++;
        }
        Debug.logInfo("Timing filterByCondition with EntityUtil for " +
                      testTime + " ms: " + count, "");
```

In the folder \${OFBizInstallFolder}\runtime\logs, look into the file ofbiz.log and search for the string **Timing filterByCondition**. We will probably see something like this:

```
Timing filterByCondition with database access for 2000 ms: 1334
Timing filterByCondition with EntityUtil for 2000 ms: 2567
```

Like post-query ordering, post-query filtering is also faster (by more than two times) than doing a re-query. However, performing post-query filtering and ordering should be used carefully and it is not wise to get into the habit of performing a findAll query and filtering the returned RecordSets. Filtering large RecordSets uses a lot of memory and resources and allows the database to do the work when dealing with large amounts of data is more efficient.

Post-Query Filtering by Date

The class method `filterByDate` of the class `EntityUtil` takes in a list of `GenericValue` objects and returns a list of what is active at the moment. It uses the fields `fromDate` and `thruDate` to determine whether a record is active.

In the folder `${EntityAccessScriptDir}`, create a new file `filterByDateTiming.bsh` and enter into it this:

```
import org.ofbiz.entity.util.*;
import org.ofbiz.base.util.*;

allRecords = delegator.findAll("Planet");

conditions = EntityUtil.getFilterByDateExpr();

long testTime = 2 * 1000; // 2 seconds
long endTime = System.currentTimeMillis() + testTime;
long count = 0;
while (System.currentTimeMillis() <= endTime) {
    data = EntityUtil.filterByCondition(allRecords, conditions);
    count++;
}
Debug.logInfo("Timing filterByDate with filterByCondition for " +
    testTime + " ms: " + count, "");

endTime = System.currentTimeMillis() + 2000;
count = 0;
while (System.currentTimeMillis() <= endTime) {
    data = EntityUtil.filterByDate(allRecords);
    count++;
}
Debug.logInfo("Timing filterByDate direct for " +
    testTime + " ms: " + count, "");
```

As can be seen, the method `filterByDate` is like a short cut of:

```
conditions = EntityUtil.getFilterByDateExpr();
data = EntityUtil.filterByCondition(allRecords, conditions);
```

Since both ways of filtering involve retrieved records in memory rather than re-querying of the database, both perform about the same in terms of speed. The more direct method `filterByDate` is slighter faster, as can be seen in the folder `${OFBizInstallFolder}\runtime\logs` file `ofbiz.log`:

```
Timing filterByDate with filterByCondition for 2000 ms: 15282
Timing filterByDate direct for 2000 ms: 15710
```

The `filterByDate` method takes in an optional second parameter that is a `java.util.Date` object, which makes the method retrieve records active at some date.

Specifying Custom fromDate and thruDate Field Names

To specify custom field names for the fields `fromDate` and `thruDate`, we need to use a rather cumbersome form of the `filterByDate` method which takes in five parameters:

- `datedValues` of type `java.util.List`, which is the list of records in memory to go through.
- `moment` of type `java.sql.Timestamp`, which is the moment at which records must be active.
- `fromDateName` of type `java.lang.String`.
- `thruDateName` of type `java.lang.String`.
- `allAreSame` of type `boolean`, which specifies whether records in `datedValues` are all of the same Entity or not.

The last parameter `allAreSame` is usually true. We seldom find the need to go through a non-homogeneous list of database records of various entities.

The second parameter `moment` can be constructed from a `java.util.Date` object like this:

```
moment = UtilMisc.toDate("12/31/2007 00:00:00");
momentTs = new java.sql.Timestamp(moment.getTime());
```

A typical script that uses this form of `filterByDate` is like this:

```
import org.ofbiz.entity.util.*;
import org.ofbiz.base.util.*;

allRecords = delegator.findAll("MyPlanet");

date = UtilDateTime.toDate("12/30/2007 00:00:00");
data = EntityUtil.filterByDate
(allRecords, new java.sql.Timestamp(date.getTime()),
 "myFromDate", "myThruDate", true);
```

Other Post-Query Filtering Methods

`EntityUtil` also has other filtering methods, all class methods.

The `filterByAnd` Method

This is analogous to the instance method `findByAnd` of the class `GenericDelegator`. It also takes in a `java.util.Map` for its second parameter, just like the method `findByAnd` does.

The filterByOr Method

This is analogous to the instance method `findByOr` of the class `GenericDelegator`. It does not take a `java.util.Map` for its second parameter, unlike the method `filterByAnd`.

Using the Entity Engine Cache

OFBiz has a really neat feature that, if used correctly, can drastically increase the application's performance by dramatically cutting down the number of queries sent to the database. It does this by using what is known as the Entity Engine Cache. The principle of a cache is to act as an easily accessible store of data. This store is first checked and if the required data is there then it is retrieved immediately. If the required data is not present in the cache then it is obtained, in the case of the Entity Engine Cache, via a database lookup. The data is then placed into the cache. The next time the same information is required, the cache is again checked and the data is returned immediately from the cache.

Since the cache is held in memory, the lookup is now performed much faster.

Using the cache is easy. Where we would normally use the `GenericDelegator` object's `findByPrimaryKey()` method to return one record, we use `findByPrimaryKeyCache()`. The Cache appendage generally applies to any of the lookup methods (for example, `findAll()` and `findAllCache()`) allowing us to not only store single `GenericValue` records in the cache, but also multiple records (Lists of `GenericValues`).

Using the cache correctly is of the utmost importance.

The size of the cache is finite. It is dependent on the memory settings and the amount of memory allocated to OFBiz. Once the cache is full, the oldest records are garbage collected and new ones can be added. This is where the danger comes in. Imagine a large organization with tens, or even hundreds of thousands of customers.

As an example, Writing code that placed every single Party record into the Cache looks like this:

```
List parties = delegator.findAllCache("Party");
```

Every time someone entered the Party Manager application and performed a lookup with no parameters, it would empty the cache of important, much needed data which would then have to be re-cached. The cache is most effective when it is filled with important, frequently used data.

Examples of where the Entity Engine Cache is best used are seen in the `ProductStore` and `ProductCategory` entities. The `ProductStore` entity for most organizations would normally only contain just a few records. This entity contains store-wide information and is used frequently throughout the ecommerce application. Since this ecommerce application is the customer facing application, this is where we would expect to see the highest level of traffic. Information contained in fields of this `ProductStore` record are used to determine the functionality, look and feel and to a certain extent the flow between pages (whether the cart should be displayed every time an item is added) of the ecommerce application. The cache is not just session-wide. When the first person enters the first page of the ecommerce application, the database is queried and the record is returned. From then on, the record is pulled from the cache by other users as well as our initial visitor, saving potentially millions of single record lookup queries. Likewise, the category structure is also cached. Imagine the database traffic and amount of queries if the category structure had to be built from data in the database every time a page was visited!

The only downside to this is that when we decide to update the `ProductStore` record or perhaps decide to rearrange our category structure , before changes are visible to the ecommerce application, the cache must first be cleared. This is as simple as clicking the **Clear All Caches** link from the **Cache Maintenance** screens in the **Webtools Application**.

Dynamic View Entities

It may sometimes be the case that the complexity of the query we are using to retrieve data from the database is dependent on a set of parameters. Joins can be expensive in terms of database performance, particularly when we are joining two entities that can potentially hold millions of records. We may wish to use the same XHTML form and the same piece of code to perform a lookup, yet alter which entities are included in the lookup.

This is best explained by example. Head to the Order Manager Application by firing an OFBiz http request to webapp `ordermgr` and log in using the standard **ofbiz** user login. Select **Find Orders**.

We can search for orders using a number of different parameters. For now, let's concentrate on just three fields: The **Order Id**, the **ProductId**, and the **Status Id**. Enter our pre-placed `orderId` of **WS10000** into the **Order Id** field, select **Completed** as the **Status Id** and tick the **show all records** box. We can see that our order has been found.

Now empty the orderId field and enter **GZ-2644** into the **Product Id** field. This is the `productId` for the **Round Gizmo** we ordered in Chapter 1. Again our order is returned. Although you would not have noticed the difference, the second lookup was more expensive than the first because an extra Entity was included in the lookup. This means an extra SQL `JOIN` was added to the query. The `OrderHeader` Entity stores the single record that is our order. It contains information that is order-wide and tells us what type of order it is, when it was placed, its status, how it was placed, and its unique identification number. When we are searching on just the `orderId` and the `statusId`, querying just the `OrderHeader` entity is enough. However, when we need to find all orders containing a certain product, looking at just the `OrderHeader` entity is no longer enough. Since, the `OrderHeader` has no `productId` field we must examine the `OrderItem` entity. We have also specified we wish to show only orders with the status of **Completed**, and the `statusId` is a field on the `OrderHeader`, so we definitely have to perform a join. While the database is small these joins are instantaneous, whereas when we are dealing with millions of `OrderHeader` records, and many millions of `OrderItem` records, performing these unnecessary joins can be highly wasteful. For the sake of the explanation this has been simplified, we could be talking about joining unnecessarily across five or more entities when all the information could have been returned from just one.

We could create an endless amount of **static** View Entities by adding their structures to the `entitymodel.xml` files for every possible combination of joins we need to perform. However, this would quickly get out of hand, and we could end up with hundreds of extra entities which would be difficult to manage. The best solution is to use a Dynamic View Entity to programmatically create complex queries on the fly.

We can recreate all of the functionality that we are given by the regular static View Entities in Java code and we can dynamically add relations to other Entities to make the processing of the records simpler and more efficient.

Let's start by taking a look at our Hotel example from the *View Entities* section in Chapter 7. We are going to use the playground we have been using in our above examples so we can concentrate solely on the structure of the Dynamic View Entity. In `LearningForms.xml` find the existing Form Widget named `ScriptNameForm` and replace it with:

```
<form name="ScriptNameForm" type="single"
      target="ProcessEntityAccessBSF">
<field name="scriptName"><hidden
      value="dynamicViewEntity.bsh"/></field>
<field name="hotelName"><text/></field>
<field name="ownerFirstName"><text/></field>
<field name="city"><text/></field>
<field name="Submit"><submit/></field>
</form>
```

Notice the hidden field `scriptName`. We have now hard-coded the script to be processed so we do not have to type it every time we perform a search.

Now in our `${EntityAccessScriptDir}` folder, add a new file called `dynamicViewEntity.bsh`:

```
import org.ofbiz.base.util.UtilValidate;
import org.ofbiz.base.util.UtilMisc;
import org.ofbiz.entity.model.DynamicViewEntity;
import org.ofbiz.entity.model.ModelKeyMap;
import org.ofbiz.entity.condition.*;
import org.ofbiz.entity.util.EntityFindOptions;
import org.ofbiz.entity.util.EntityListIterator;

String hotelName = (String)request.getParameter("hotelName");
String ownerLastName =
        (String)request.getParameter("ownerLastName");
String city = (String)request.getParameter("city");

List andExpr = new ArrayList();
List fieldsToSelect = UtilMisc.toList("hotelId", "hotelName",
                                         "lastName");

List orderBy = UtilMisc.toList("hotelName");

DynamicViewEntity dve = new DynamicViewEntity();
dve.addMemberEntity("HL", "HotelL");
dve.addAliasAll("HL", ""); // no prefix

if(UtilValidate.isNotEmpty(hotelName)){
    andExpr.add(new EntityExpr("hotelName", EntityOperator.EQUALS,
                               hotelName));
}

dve.addMemberEntity("OP", "PersonL");
dve.addAlias("OP", "lastName");
dve.addViewLink("HL", "OP", Boolean.FALSE, UtilMisc.toList(new
    ModelKeyMap("ownedBy", "personId")));
if(UtilValidate.isNotEmpty(ownerLastName)){
    andExpr.add(new EntityExpr("lastName", EntityOperator.EQUALS,
                               ownerLastName));
}

if(UtilValidate.isNotEmpty(city)){
    dve.addMemberEntity("PA", "PostalAddress");
```

```

        dve.addAlias("PA", "city");
        dve.addViewLink("HL", "PA", Boolean.FALSE, UtilMisc.toList(new
            ModelKeyMap("postalAddressId", "contactMechId")));
        andExpr.add(new EntityExpr("city", EntityOperator.EQUALS,
            city));
    }

    EntityFindOptions findOpts = new EntityFindOptions(true,
        EntityFindOptions.TYPE_SCROLL_INSENSITIVE,
        EntityFindOptions.CONCUR_READ_ONLY, true);
    EntityCondition cond = new EntityConditionList(andExpr,
        EntityJoinOperator.AND);

    EntityListIterator eli =
        delegator.findListIteratorByCondition(dve, cond, null,
            fieldsToSelect, orderBy, findOpts);
    data = eli.getCompleteList();
    eli.close();
}

```

To see this in action, return to the playground by returning to <http://localhost:8080/learning/control/ProcessEntityAccessBSF>.

Leaving all of the fields blank, click **Submit**:

Hotel Name	<input type="text"/>
Owner Last Name	<input type="text"/>
City	<input type="text"/>
Submit	

Processed script: "dynamicViewEntity.bsh"

lastName	hotelId	hotelName
Pitt	1	Brad's Hotel
Hanks	2	Hank's Hotel

Paying careful attention to the highlighted code in the code example above, we can see that the `Hotell` entity was added to the `DynamicViewEntity` as a Member Entity, as was the `PersonL` entity regardless of the input parameters. `PostalAddress` on the other hand, is only added when the `city` variable is not empty, that is, when the `city` parameter is passed through from the Form. Now enter **New York** into the **City** field. The `PostalAddress` entity is now added to the `DynamicViewEntity` and the where condition `PostalAddress.city = 'New York'` is added to the `EntityCondition` resulting in the single hotel we know of in New York, **Hank's Hotel**, being retrieved.

Left Outer Joins with the Dynamic View Entity

From the **Entity Data Maintenance** screens in **Webtools**, create a new Hotel with a hotelId value of three and a hotelName of **Luxury Hotel**. We do not want to populate the postalAddressId field or the ownedBy fields, since we are not sure where it is or who owns it for the sake of this example.

Return to our playground Script Processor page and again leaving all fields blank, press **Submit**. Again only two fields are retrieved. The lookup performs an **INNER JOIN** across two entities, the **HotelL** and the **PersonL** and the **JOIN** is performed on the **ownedBy** field and the **personId**. Since we do not know who owns the **Luxury Hotel** we have just added, the **ownedBy** field is null and the record is not included in the **INNER JOIN**.

Examining exactly how the **PersonL** entity is added to the **DynamicViewEntity** we can see that:

```
dve.addViewLink("HL", "OP", Boolean.FALSE, UtilMisc.toList(new  
    ModelKeyMap("ownedBy", "personId")));
```

The **addViewLink** method takes the parameters **HL** which is the **HotelL** Entity alias, **OP** which is the Entity alias we have assigned to the **PersonL** entity, **Boolean.FALSE**, which states whether this relation is not optional and finally the **ModelKeyMap** which states that the **INNER JOIN** is performed on **HotelL.ownedBy = PersonL.personId**.

By making the third parameter **Boolean.TRUE** we can make this relation optional. This will perform a **LEFT OUTER JOIN** between the **HotelL** and the **PersonL** entities.

In the code change the above line to:

```
dve.addViewLink("HL", "OP", Boolean.TRUE, UtilMisc.toList(new  
    ModelKeyMap("ownedBy", "personId")));
```

and hit **F5** to refresh and perform another lookup.

lastName	hotelId	hotelName
Pitt	1	Brad's Hotel
Hanks	2	Hank's Hotel
	3	Luxury Hotel

Now the **Luxury Hotel** record is returned, even though we don't know who owns it.

Performing the Lookup

Notice the line of code that actually performs the lookup:

```
EntityListIterator eli =
    delegator.findListIteratorByCondition(dve, cond, null,
    fieldsToSelect, orderBy, findOpts);
```

This GenericDelegator takes some parameters we have already met including the DynamicViewEntity itself, the WHERE EntityCondition, the HAVING EntityCondition (null in our example), the fields we wish to return and the order we wish to return them in.

EntityFindOptions

Finally we have a parameter that is unfamiliar to us. The `findOpts`. This `EntityFindOptions` object contains variables used to select advanced finding options including instructions on how the `ResultSet` can be traversed or if it can be updated, whether we wish to return a distinct (perform a `SELECT DISTINCT` query) or `LIMIT` the number of records returned.

Take a look at the `org.ofbiz.entity.util.EntityFindOptions` class for all of the methods and settings available for us to use in the lookup.

In the above example we can see that our `EntityFindOptions` constructor sets all the options we need:

```
EntityFindOptions findOpts = new EntityFindOptions(true,
    EntityFindOptions.TYPE_SCROLL_INSENSITIVE,
    EntityFindOptions.CONCUR_READ_ONLY, true);
```

Equates to the constructor:

```
EntityFindOptions (boolean specifyTypeAndConcur, int resultSetType,
    int resultSetConcurrency, boolean distinct)
```

Since the first parameter is `true`, the following two parameters (`resultSetType` and `resultSetConcurrency`) are used to specify how the results will be used. If this value is `false`, the default values for the JDBC driver are used. Since the `resultSetType` parameter is set to `Scroll Insensitive`, we are able to scroll in either direction or to move the cursor to a particular row. The `resultSetConcurrency` is set to `Concur Read Only`. A `ResultSet` that is set to `Read Only` does not allow its contents to be updated. Finally the `distinct` parameter is set to `true` specifying that the values returned should be filtered to remove duplicate values.

Paginating Using the EntityListIterator

Finally we come to the results that are actually returned from the lookup. Although we may be expecting a List of values to be returned, it is not a List we get. It is an EntityListIterator. This EntityListIterator is a cursor List Iterator for handling censored database results. We can use this object to navigate through the ResultSet, scrolling forwards and (since we have set the resultSetType to ScrollInsensitive) backwards, returning the current record. One useful function of the EntityListIterator is to be able to return a partial list of results from the ResultSet by using the method `getPartialList(int start, int number)` which will return a number of results from the absolute position denoted by the start parameter.

In the above example we simply used the `getCompleteList()` method to return the entire ResultSet as a List. For our simple example, where we are dealing with only two or three records, this is acceptable. However, returning the entire ResultSet in one go for thousands or even perhaps millions of records will have serious memory implications. By using the `getPartialList` method we are able to only return a set number of records, and since we are able to specify the start position in the RecordSet, we are able to paginate through the entire RecordSet.

[ The most important thing when dealing with an EntityListIterator is to remember that it must be closed. An unclosed EntityListIterator, whether it is unclosed due to an error during the iteration or because it was not explicitly closed using the `close()` method will result in Out of Memory Errors.]

Paginating through Party Records: A Working Example

The best way to see how we can paginate through a Recordset is to see one of the working examples already within the core code. With OFBiz started open the **Party Manager Application** by firing an http request to webapp partymgr. The main screen in the **Party Manager Application** is the **Find Party** screen. From here click on the **Show all records** link. A list of 20 records will be displayed to the screen. Looking at the top right of the **Parties Found** box, we can see that we are looking at **1 - 20 of 52** records. (The total number of records returned may well be different depending on how many Parties have been added to your local database).

Parties Found					Previous 1 - 20 of 52 Next				
Party ID	User Login	Name	Type		Details	Orders	Quotes	New Order	New quote
NA	(None)	(No name found)	Person		Details	Orders	Quotes	New Order	New quote
admin	(Many)	ADMINISTRATOR, THE	Person		Details	Orders	Quotes	New Order	New quote
system	system	Account, System	Person		Details	Orders	Quotes	New Order	New quote
itdadmin	(Many)	Administrator Limited	Person		Details	Orders	Quotes	New Order	New quote

Click **Next** and notice that we will now be looking at records **21 - 40 of 52**. We have paginated to the next 20 records in the RecordSet.

We are going to examine exactly how this pagination has worked by extracting all of the relevant code snippets from all of the relevant files and Java classes.

Let's start with FreeMarker code that outputs the above navigation bar, presenting us with the Next and Previous buttons. From the file \${webapp:partymgr}\party\findparty.ftl at line 163 we can see:

```

<div class="screenlet-title-bar">
    <ul>
        <h3>${uiLabelMap.PartyPartiesFound}</h3>
        <#if (partyListSize > 0)>
            <#if (partyListSize > highIndex)>
                <li><a class="nav-next" href="<@ofbizUrl>
                    findparty?VIEW_SIZE=${viewSize}&
                    VIEW_INDEX=${viewIndex+1}&
                    hideFields=${parameters.hideFields?
                    default("N")}${paramList}<@ofbizUrl>">
                    ${uiLabelMap.CommonNext}</a></li>
            <#else>
                <li class="disabled">${uiLabelMap.CommonNext}</li>
            </#if>
            <li>${lowIndex} - ${highIndex} ${uiLabelMap.CommonOf}
                ${partyListSize}</li>
            <#if (viewIndex > 0)>
                <li><a class="nav-previous" href="<@ofbizUrl>
                    findparty?VIEW_SIZE=${viewSize}&
                    VIEW_INDEX=${viewIndex-1}&
                    hideFields=${parameters.hideFields?
                    default("N")}${paramList}<@ofbizUrl>">
                    ${uiLabelMap.CommonPrevious}</a></li>
            <#else>
                <li class="disabled">${uiLabelMap.CommonPrevious}</li>
            </#if>
            </#if>
        </ul>
        <br class="clear"/>
    </div>

```

The highlighted parts of this code display the **Next** button, the **X - Y of Z** records and finally the **Previous** button.

The location of the links behind both the **Next** and **Previous** buttons are almost the same:

```
<@ofbizUrl>findparty?VIEW_SIZE=${viewSize}&VIEW_INDEX=${viewIndex+1}&hideFields=${parameters.hideFields?default("N")}${paramList}</@ofbizUrl>
```

We will be exploring the exact meaning of the `<@ofbizUrl>` tags in the section called *OFBiz Transform Tags* in Chapter 14. For now all we need to know is this link will resolve to:

```
/findparty?VIEW_SIZE=${viewSize}&VIEW_INDEX=${viewIndex+1}&hideFields=${parameters.hideFields?default("N")}${paramList}
```

There are a few variables that are used in the pagination. They are:

- `viewSize`—This is the amount of records we wish to display, in this case, 20.
- `viewIndex`—This is the index or page we are on. The first 20 records has a `viewIndex` of 0, records 21–40 has a `viewIndex` of 1 and so on. Notice how the Previous link simply decrements this value: `VIEW_INDEX=${viewIndex-1}`, while the Next button increments it: `VIEW_INDEX=${viewIndex+1}`.
- `lowIndex`—This is the lowest absolute position of the cursor in the RecordSet. In the above picture the `lowIndex` value is 1.
- `highIndex`—This is the highest absolute position of the cursor in the RecordSet. In the above picture the `highIndex` value is 1.
- `partyListSize`—This is the total number of records in the entire RecordSet.

The `paramList` variable simply contains a full List of all of the user entered search criteria. For example, had we entered a **Last Name** as **Smith** in the search fields, this variable would contain `lastName=Smith`.

The variable `hideFields` states whether or not to display the search fields and has nothing to do with the actual pagination.

So, we can see that clicking on the **Next** button will pass the `VIEW_SIZE` and `VIEW_INDEX` parameters, along with any other parameters in the `paramList`, through to the `findparty` request. We can see that `request-map` for `findparty` does not call a service as an event as we may expect, it simply passes us through to the `findparty` `view-map` which in turn passes us immediately to the `findparty` Screen Widget in `${ofbizComponent:party}\widget\partymgy\PartyScreens.xml`.

Locate the code for this Screen Widget and we can see where the lookup service is invoked from.

```
<service service-name="findParty" auto-field-map="parameters"/>
```

Sure enough, if we examine the service definition for the `findParty` service (located in `party\servicedef\services_view.xml`) we can see that the GET parameters `VIEW_INDEX` and `VIEW_SIZE` are not compulsory for this service, but they are certainly expected, along with all of the other search criteria parameters (for example `lastName`):

```
<attribute name="VIEW_INDEX" type="String" mode="IN"
           optional="true"/>
<attribute name="VIEW_SIZE" type="String" mode="IN" optional="true"/>
```

We can also see from this service definition that the `findParty` method that is invoked resides in the class `org.ofbiz.party.party.PartyServices`.

Open this class and see at line 996 the `VIEW_INDEX` and `VIEW_SIZE` parameters being obtained:

```
// set the page parameters
int viewIndex = 0;
try {
    viewIndex = Integer.parseInt((String)
        context.get("VIEW_INDEX"));
} catch (Exception e) {
    viewIndex = 0;
}
result.put("viewIndex", new Integer(viewIndex));

int viewSize = 20;
try {
    viewSize = Integer.parseInt((String)
        context.get("VIEW_SIZE"));
} catch (Exception e) {
    viewSize = 20;
}
result.put("viewSize", new Integer(viewSize));
```

and at line 1018 the `partyListSize`, `lowIndex` and `highIndex` variables being declared:

```
int partyListSize = 0;
int lowIndex = 0;
int highIndex = 0;
```

Next follows the creation of the `DynamicViewEntity`. We now have enough knowledge to understand this code in full. Follow it through seeing which entities are included in the creation of the Dynamic View. Notice the differences now that the `DynamicViewEntity` is being created in the Java code, in particular the use of the `try / catch` blocks around the lookup.

The actual lookup is performed on line 1292. Let's take a look at the code that performs the lookup and examine what is happening line by line. First we perform the lookup returning out `EntityListIterator pli`.

```
EntityListIterator pli = delegator.findListIteratorByCondition  
    (dynamicView, mainCond, null, fieldsToSelect, orderBy, findOpts);
```

Next the values for the `lowIndex` and `highIndex` are calculated. Imagine that we are looking at 20 records per page. On the second page (that is `viewIndex=1`) our `lowIndex` would equal 21 and our `highIndex` would equal 40:

```
lowIndex = viewIndex * viewSize + 1;  
highIndex = (viewIndex + 1) * viewSize;
```

Cycle through the `RecordSet` starting at the value just calculated for `lowIndex`. For example, starting at the 21st record in the `RecordSet`, get the next 20 records and put them into a `List` called `partyList`:

```
partyList = pli.getPartialList(lowIndex, viewSize);
```

We now wish to obtain the size of the `RecordSet`. The most efficient way of doing this is to move the cursor to the very last record and return the absolute position of this record:

```
pli.last();  
partyListSize = pli.currentIndex();
```

The final page in the pagination may have a `highIndex` value that has been calculated that is actually higher than the number of records in the `RecordSet`. For example, paginate to the third and final page of the Party list. The `highIndex` calculated should be 60. The next piece of code readjusts this value to the total number of records, so we get **41 - 52 of 52** rather than the odd looking **41 - 60 of 52**.

```
if (highIndex > partyListSize) {  
    highIndex = partyListSize;  
}
```

We have finished with the `EntityListIterator` so it must be closed to save resources.

```
pli.close();
```

Finally before returning from this service on line 1321, the variables must all be added to the result Map and passed out of the service as OUT parameters and picked up again back in the `findparty.ftl`

```
result.put("partyList", partyList);
result.put("partyListSize", new Integer(partyListSize));
result.put("paramList", paramList);
result.put("highIndex", new Integer(highIndex));
result.put("lowIndex", new Integer(lowIndex));
```

Functions and Dynamic View Entities

Dynamic View Entities are incredibly versatile. In fact, ultimately, they use the same underlying code to build the Model Entity as static or normal View Entities. As such, we can recreate any of the functionality dynamically that is available to us using normal view entities, including the use of functions. This however can quickly become complicated and its use is not widespread. There is only one real example of the addition of a Complex Alias (see Chapter 7) to a Dynamic View Entity in Java code and this can be found in the class `org.ofbiz.product.product.ProductSearch` where the relevancy of a number of keywords is calculated dynamically on line 278.

A simpler and clearer way to perform functions like `sum` and `avg` is simply to create a normal View Entity as we did in Chapter 7 and add this View Entity as a Member Entity to our `DynamicViewEntity` exactly as if it were a normal Entity. Remember when creating a View Entity to perform a function that a View Entity does not necessarily have to join two entities. A View Entity can have just one Member Entity.

Summary

We have covered a lot of ground in this chapter and should now be able to comfortably understand much of the Java code in the applications components. We have spent some time unraveling some of the mysteries of the Entity Engine and learnt how to use some of its features to improve the application's performance. In this chapter we have studied:

- How to create, retrieve, update and delete database records.
- How to use `GenericDelegator`'s instance method `findByCondition` to:
 - Retrieve database records by conditions (second parameter of method).
 - Select subset of record fields to be retrieved (third parameter, or fourth if using `HAVING` clause).

- Order records to be returned (fourth parameter, or fifth if using `HAVING` clause).
 - Retrieve summary records by a `HAVING` clause (third parameter).
- OFBiz's implementation of conditions (`EntityCondition` objects, created with `EntityComparisonOperator` objects).
- How to combine multiple conditions into complex conditions (with `EntityJoinOperator` objects).
- Shortcuts for common styles of conditions, using `GenericDelegator`'s instance methods `findByAnd` and `findByOr`.
- How to count the number of records satisfying a set of conditions, using `GenericDelegator`'s instance methods `findCountByAnd` and `findCountByCondition`.
- OFBiz's way of dealing with expirable database records and with maintaining audit trails – with fields `fromDate` and `thruDate`.
- How to get related records for any database record we retrieve, using `GenericValue`'s instance methods `getRelated` and `getRelatedOne`.
- Various forms of post-query processing of retrieved database records, needed at times to avoid expensive re-querying of database (`EntityUtil`'s class methods).
- How to improve performance by using the Entity Engine Cache and implications of using the cache incorrectly.
- How to create complicated queries using programmatically using Dynamic View Entities.
- How to paginate through large RecordSets using the `EntityListIterator` and taken a closer look at how all lookups are performed in OFBiz.

We have now covered the basics of all 3 parts of the MVC framework: the "Model", "View" and the "Controller". In the next chapter, we will be revisiting the Controller, examining how we can create some complex business logic to control the flow through our applications.

9

The Events

We have understood the basics of all of the different parts of the OFBiz MVC framework, we now return to the Controller part and explore the meatier part of the flow – the programming of business logic.

In Chapter 6 we merely covered the networking or communications channels (flow) of an application. This chapter deals with the actual working of an application, such as whether a loan application should be accepted or rejected and on what basis, whether a purchase qualifies for a refund, and any other logic to do with the running of a business or organization.

In this chapter, we will be looking at:

- Java events and their context
- OFBiz-related techniques for programming business logic
- Security-related programming
- Handling parameters sent by an end-user
- Sending feedback to an end-user
- Getting our application to speak in various languages
- Database-related programming (quick recap of Chapter 8)

The context for OFBiz "service" events will be discussed in detail in the next chapter.

Java Events

Although we have already seen a couple of examples of simple Java events, we have not yet seen how to perform any useful business logic. The OFBiz framework places a number of useful objects onto the request as attributes for us. Other useful objects can be taken from the session. These objects are available for us to use from within our Java method, once we have extracted them from the right places.

As you will recall, the request is passed into the Java event as a parameter. The session can be obtained from the request in the standard way.

```
HttpSession session = request.getSession();
```

The availableLocales object can be constructed like this:

```
List UtilAvailableLocales = UtilMisc.availableLocales();
```

The locale object can be retrieved like this:

```
Locale locale = UtilHttp.getLocale(request);
```

The delegator object can be retrieved like this:

```
GenericDelegator delegator =
(GenericDelegator)request.getAttribute("delegator");
```

The dispatcher object can be retrieved like this:

```
LocalDispatcher dispatcher =
(LocalDispatcher)request.getAttribute("dispatcher");
```

The security object can be retrieved like this:

```
Security security = (Security)request.getAttribute("security");
```

The userLogin object can be retrieved from the session:

```
GenericValue userLogin =
(GenericValue)session.getAttribute("userLogin");
```

Security and Access Control

So far, our scripts and events have not been bothered with access control; any user can access them. We will now explore the security and access control mechanisms in OFBiz.

We will be using a Person record in OFBiz with first name OFBiz and last name Researcher. This Person record was created in Chapter 1 when we created an anonymous shopper to buy four "Round Gizmos". If the Person record doesn't exist for you, you can create a new Person through the "Party Manager" application. Go to <http://localhost:8080/partymgr> select Create and Create New Person.

User Logins are like Access Cards

A userLogin in OFBiz (or any software with access control) is like an access card that a person carries to gain access into some or all parts of a secure building. One or more userLogins can be created for a person, with each userLogin allowing access to different areas.

We can see that when we take the userLogin from the request, the object we get is a GenericValue. This GenericValue relates to a record in the UserLogin entity. Open up the **Entity Data Maintenance** screen for UserLogin and find the record with a userLoginId of admin. The userLogin has a foreign key relationship partyId to the Party entity. The partyId is the user's (whether they be a Person or a PartyGroup) unique identification number, this is one of the most widely used and important IDs across all of the OFBiz components.

So, to obtain the Party record from the userLogin we would have:

```
GenericValue userLogin =
    (GenericValue) session.getAttribute("userLogin");
GenericValue party = null;
Try{
    party = userLogin.getRelatedOne("Party");
}catch(GenericEntityException e){
    Debug.logError(e, module);
}
```

From the Party we can then determine whether they are a Person or a PartyGroup (Organization).

Once logged in, the user carries around the information about who they are, everywhere they go.

The Events

In the **Party Manager Application**, bring up our Person record.

Profile	Role(s)	Link Party	Relationships	Communications	Vendor	Tax Infos	Rates
Shopping Lists							
Segments							
Classifications							
Contact List							
Party Content							
Billing Acct	Orders	Quotes	Requests	New Order	New quote	Payments Sent	
Payments Received	Financial Accounts						

The Profile of OFBiz Researcher [10000]

[Show Old](#)

Personal Information | [Update](#)

Name OFBiz Researcher

Contact Information | [Create New](#)

Contact Type	Contact Information	Soliciting OK?	
Phone Number	Main Home Phone Number 1 2-3 (Updated: 2008-05-26 18:44:25.89) <input type="button" value="Request For Bug Fix"/>	0	Update Expire
Email Address	Primary Email Address someaddr@somewhere.com (send email) (Updated: 2008-05-26 18:44:25.906) <input type="button" value="Request For Bug Fix"/>	0	Update Expire
Postal Address	Billing (AP) Address Shipping Destination Address To: Gift Recipient Somewhere A City, AK 123456 United States (Updated: 2008-05-28 18:48:01.109) <input type="button" value="Request For Bug Fix"/>	0	Update Expire

Scroll down to section labeled **User Name(s)**.

User Name(s)	Create New
---------------------	----------------------------

Click **Create New** to create a userLogin. Create two userLogins with user login IDs **allowed** and **denied**, respectively. Set the password to **ofbiz** for both. Enter **zibfo** for **Password Hint**.

The Profile of OFBiz Researcher [10000]

Create UserLogin

User Login Id	<input type="text"/>
Current Password	<input type="password"/>
Current Password Verify	<input type="password"/>
Password Hint	<input type="text"/>
<input type="button" value="Save"/> <input type="button" value="Cancel/Done"/>	

User Name(s)		Create New
User Login	allowed ENABLED	Edit Security Groups
User Login	denied ENABLED	Edit Security Groups

Security Groups are like Access Levels

A security group is a single group of access rights. There can be many ways of defining security groups, depending on how we want to structure our access control. One way is the familiar "clearance level" mentioned in many spy movies—different areas in a building are assigned different clearance levels say 1 to 10 (10 being most classified and requiring highest access privileges). Another way is via the use of "color codes", each color representing a single department—this would serve to confine employees to their own departments.

From the **Party Manager Application**, click on **Security** then **New Security Group**. Create a new security group with an ID of **LEARNSCREENS** and description **Learning Screens**.

Edit Security Group With ID []

[New Security Group](#)

Security Group ID	<input type="text" value="LEARNSCREENS"/>
Description	<input type="text" value="Learning Screens"/>
<input type="button" value="Update"/>	

Security Permissions are like Individual Secured Areas

A security permission is a single unit of access and is the smallest indivisible unit of security. It is like a specific entry code for a single secured room in a building.

Unfortunately, OFBiz doesn't provide a ready-made screen and form for the creation (nor modification) of security permissions. Go to the **Entity Data Maintenance** screen for entity **SecurityPermission**, and create a new **SecurityPermission** record with **permissionId** of **LEARN_VIEW** and **description** containing **Viewing of Learning Screens**.

permissionId	LEARN_VIEW
description	Viewing of Learning Screens

Security Permissions are Contained within Security Groups

It is not possible to assign individual security permissions to user logins. Only security groups can be handled. Hence, security permissions need to be contained within security groups.

Return back to the Party Manager Application's Security Groups by clicking on **Security** and find the newly added **LEARNSCREENS** group.

Permissions for SecurityGroup with ID [LEARNSCREENS]

[New Security Group](#)

Permission ID **Add Permission (from list) to SecurityGroup**

Permission ID	[LEARN_VIEW] View of Learning Screens
----------------------	---------------------------------------

Add

Add Permission (manually) to SecurityGroup

Permission ID	LEARN_VIEW
----------------------	------------

Add

Add the security permission **LEARN_VIEW** to the security group **EARNSCREENS**. Either add via the drop-down box, or manually, as shown above.

Confirm that the security permission was successfully added to the security group.

Permission ID	
[LEARN_VIEW] View of Learning Screens	Remove

User Logins are Assigned Security Groups

Go back to our Person record, section **User Name(s)**. Click on the **Security Groups** button of user login **allowed** to bring up the security groups assignment screen. Select the Group **LEARNSCREENS** and click **Add**.

The Profile of OFBiz Researcher [10000]

Edit UserLogin Security Groups

Group	From Date	Thru Date - Update	Delete Link
Add UserLogin to Security Group			
Group	[LEARNSCREENS] Learning Screens		
From Date & Time	<input type="text"/>		
Thru Date & Time	<input type="text"/>		
Add			

Confirm that the userLogin was successfully assigned to the security group.

Group	From Date	Thru Date - Update	Delete Link
Learning Screens [LEARNSCREENS]	2008-06-08 15:35:50.671	<input type="text"/>	Update Remove

Dealing with Security in Java

Having set up the required security permissions and groups and user logins, we now look at how to deal with these objects in Java.

The Events

Insert into the file \${component:learning}\widget\learning\LearningScreens.xml a new Screen Widget:

```
<screen name="CheckAccess">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="title">
                    <label text="Checking Access via Event BeanShell"/>
                </decorator-section>
                <decorator-section name="body">
                    <label text="${eventMessageList[0] }"/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

Insert into the file \${webapp:learning}\WEB-INF\controller.xml a new request map:

```
<request-map uri="CheckAccess">
    <security auth="true"/>
    <event type="java"
        path="org.ofbiz.learning.learning.LearningEvents"
        invoke="checkAccess "/>
    <response name="success" type="view" value="CheckAccess"/>
</request-map>
```

and a new view map:

```
<view-map name="CheckAccess" type="screen"
    page="component://learning/widget/learning/
    LearningScreens.xml#CheckAccess" />
```

In the class org.ofbiz.learning.learning.LearningEvents, create a new static method checkAccess :

```
public static String checkAccess(HttpServletRequest request,
    HttpServletResponse response) {
    Security security = (Security)request.getAttribute("security");
    String key = "_EVENT_MESSAGE_";
```

```
if (security.hasPermission("LEARN_VIEW", request.getSession()))
{
    request.setAttribute(key, "You have access!");
}
else {
    request.setAttribute(key, "You DO NOT have access! You are
denied!");
}
return "success";
}
```

also ensure that the correct class has been imported:

```
import org.ofbiz.security.Security;
```

Stop, recompile, and restart. Fire to webapp learning an http OFBiz request CheckAccess. Login as **denied** (password **ofbiz**) and see the following screen stating denied access.



Now, logout and then fire the same request again. This time, login as **allowed** and see the following screen stating access granted.



Sending Feedback to the End-User

Part of the job of the Controller component of MVC, after performing all the work it is asked to do by the end-user, is deciding what to tell the end-user. The feedback can be to notify about the success of one or more operations, or to warn about errors.

In the previous section, we had set into the http request object an attribute of key `_EVENT_MESSAGE_`.

Conventions for Message Placeholders

OFBiz considers a few key names to be placeholders for feedback messages:

- `_EVENT_MESSAGE_`
- `_EVENT_MESSAGE_LIST_`
- `_ERROR_MESSAGE_`
- `_ERROR_MESSAGE_LIST_`
- `_ERROR_MESSAGE_MAP_`

The Widget Engine (the `org.ofbiz.widget.screen.ScreenWidgetViewHandler` handler, specifically) consolidates all messages in the above placeholders (except `_ERROR_MESSAGE_MAP_`) into two lists: `eventMessageList` and `errorMessageList`. The lists are then placed into the widget environment's `context` object.

The placeholder `_ERROR_MESSAGE_MAP_` is mostly used by the Service Engine (discussed later in the book), and is not handled by the Widget Engine.

Testing the Conventions

Thanks to the file `${component:common}\webcommon\includes\messages.ftl` which has been added to our screens by the `GlobalDecorator` we do not have to set up any other code to display our messages. This file takes care of analyzing each of the place holders and displaying them accordingly. For this reason, in the previous example, the returned message was displayed twice, once within the `messages.ftl` file and again in the Screen Widget:

```
<label text="${eventMessageList[0]}"/>
```

Insert into `${component:learning}\widget\learning\LearningScreens.xml` a new Screen Widget:

```
<screen name="FeedbackMessages">
  <section>
    <widgets>
      <decorator-screen name="CommonLearningDecorator">
        <decorator-section name="title">
          <label text="Exploring all placeholders for feedback
                     messages"/>
        </decorator-section>
        <decorator-section name="body">
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

Insert into the file `${webapp:learning}\WEB-INF\controller.xml` a new request map:

```
<request-map uri="FeedbackMessages">
  <event type="java"
        path="org.ofbiz.learning.learning.LearningEvents"
        invoke="feedbackMessages"/>
  <response name="success" type="view" value="FeedbackMessages"/>
</request-map>
```

and a new view map:

```
<view-map name="FeedbackMessages" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#FeedbackMessages"/>
```

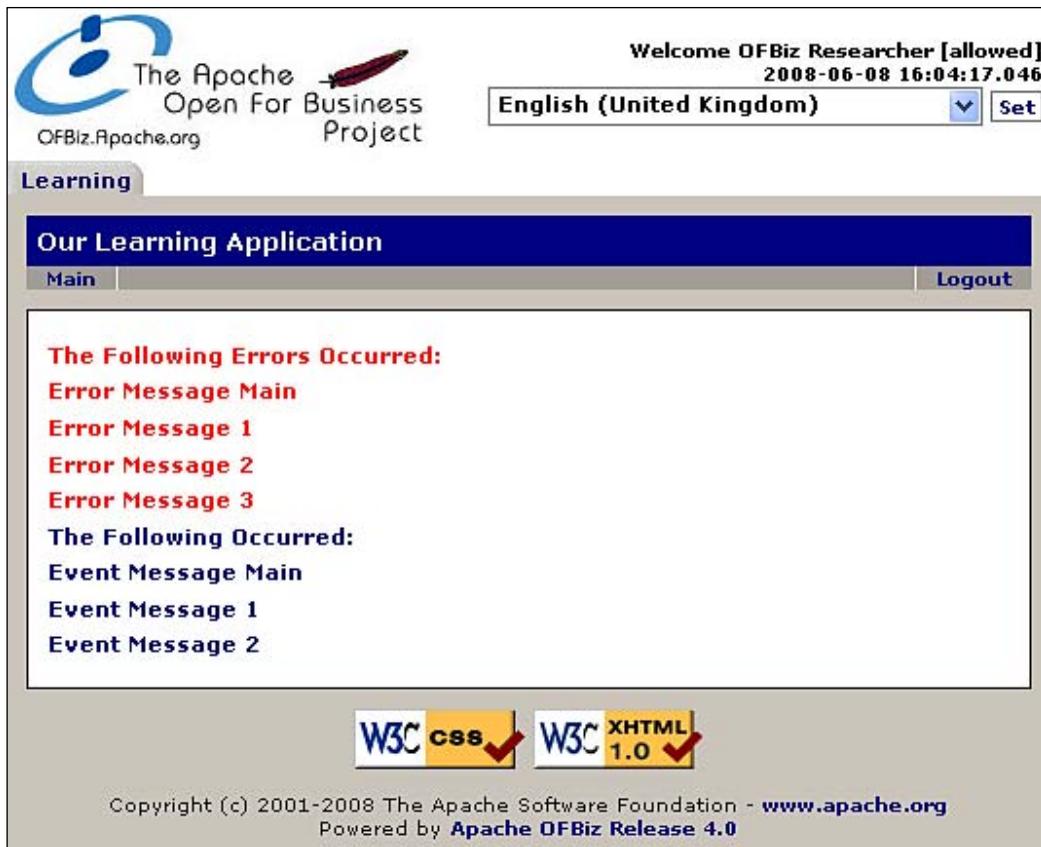
In the class `org.ofbiz.learning.learning.LearningEvents`, create a new static method `feedbackMessages`:

```
public static String feedbackMessages(HttpServletRequest request,
    HttpServletResponse response) {
    request.setAttribute("_EVENT_MESSAGE_", "Event Message Main");
    request.setAttribute
        ("_EVENT_MESSAGE_LIST_",
         UtilMisc.toList("Event Message 1", "Event Message 2"));
    request.setAttribute("_ERROR_MESSAGE_", "Error Message Main");
    request.setAttribute
        ("_ERROR_MESSAGE_LIST_",
         UtilMisc.toList("Error Message 1", "Error Message 2",
                         "Error Message 3"));
    request.setAttribute
        ("_ERROR_MESSAGE_MAP_",
         UtilMisc.toList("errMsgA", "Error Message A", "errMsgB",
                         "Error Message B"));
    return "success";
}
```

Ensure that the class `UtilMisc` has been imported:

```
import org.ofbiz.base.util.UtilMisc;
```

Shutdown, recompile the **Learning Component**, and restart. Fire to webapp `learning` an http OFBiz request `FeedbackMessages` to see how the Widget Engine consolidates feedback messages from the various placeholders into `eventMessageList` and `errorMessageList`.



Notice that the placeholder `_ERROR_MESSAGE_MAP_` was not handled at all.

Handling Parameters

As has already been seen in previous sections, sending feedback to end-users (or sending values back to View component or front-end) is done by setting attributes in the request object.

Handling parameters sent by end-users is done with the `org.ofbiz.base.util.UtilHttp` class.

In the class `org.ofbiz.learning.learning.LearningEvents`, create a new static method `receiveParams` and enter into it this:

```
public static String receiveParams(HttpServletRequest request,
HttpServletResponse response) {

    Map parameters = UtilHttp.getParameterMap(request);
    String name = "";
    String firstName = (String)parameters.get("firstName");
    String lastName = (String)parameters.get("lastName");
    if (firstName != null) name += firstName;
    if (lastName != null) {
        if (name != null) name += " ";
        name += lastName;
    }

    String message = "Welcome, " + name + "!";
    request.setAttribute("_EVENT_MESSAGE_", message);
    return "success";
}
```

Ensure that the UtilHttp class has been imported:

```
import org.ofbiz.base.util.UtilHttp;
```

Note the line above:

```
Map parameters = UtilHttp.getParameterMap(request);
```

That method neatly packs all parameters residing in a request object, sent by the end-user, into a map.

Insert into the file \${component:learning}\widget\learning\LearningScreens.xml a new Screen Widget:

```
<screen name="WelcomeVisitor">
    <section>
        <widgets>
            <decorator-screen name="main-decorator"
                location="${parameters.mainDecoratorLocation}">
                <decorator-section name="title">
                    <label text="Welcome Message"/>
                </decorator-section>
                <decorator-section name="body">
                    <include-form name="WelcomeVisitor"
                        location="component://learning/widget/learning/
                            LearningForms.xml"/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>
```

Insert into the file \${component:learning}\widget\learning\LearningForms a new Form Widget:

```
<form name="WelcomeVisitor" type="single" target="ReceiveParams">
    <field name="firstName"><text/></field>
    <field name="lastName"><text/></field>
    <field name="submit"><submit/></field>
</form>
```

In the file \${webapp:learning}\WEB-INF\controller.xml, insert a new request map:

```
<request-map uri="ReceiveParams">
    <event type="java"
        path="org.ofbiz.learning.learning.LearningEvents"
        invoke="receiveParams"/>
    <response name="success" type="view" value="WelcomeVisitor"/>
</request-map>
```

and a new view map:

```
<view-map name="WelcomeVisitor" type="screen"
    page="component://learning/widget/learning/
    LearningScreens.xml#WelcomeVisitor"/>
```

Shutdown OFBiz, recompile the **Learning Component**, and restart. Fire to webapp learning an http OFBiz request ReceiveParams, and submit some name.

The Following Occurred:	
Welcome, Some Name!	
First Name	<input type="text" value="Some"/>
Last Name	<input type="text" value="Name"/>
Submit	

Accessing Localized Messages

In our LearningUiLabels.properties file we created at the end of Chapter 5, add another two entries:

```
LearningWelcomeMsg=Welcome to France!
LearningSmallTalk=The weather is nice.
```

Properties files are organized into maps. In the above example, the properties file will be loaded into a map with two entries: LearningWelcomeMsg and LearningSmallTalk.

In file \${webapp:learning}\WEB-INF\controller.xml, insert a new request map:

```
<request-map uri="WelcomeVisitor">
    <event type="java"
        path="org.ofbiz.learning.learning.LearningEvents"
        invoke="welcomeVisitor"/>
    <response name="success" type="view" value="WelcomeVisitor"/>
</request-map>
```

We are going to use the same view-map that we used in the previous example. We do however need to change the target that the Form Widget uses. In LearningForms.xml replace the line:

```
<form name="WelcomeVisitor" type="single" target="ReceiveParams">
```

with

```
<form name="WelcomeVisitor" type="single" target="WelcomeVisitor">
```

In the class org.ofbiz.learning.learning.LearningEvents, create a new static method welcomeVisitor:

```
public static String welcomeVisitor(HttpServletRequest request,
HttpServletResponse response) {

    String resource = "LearningUiLabels";
    Locale locale = UtilHttp.getLocale(request);
    String message = UtilProperties.getMessage(resource,
                                                "LearningWelcomeMsg", locale);
    String smallTalk = UtilProperties.getMessage(resource,
                                                "LearningSmallTalk", locale);

    Map parameters = UtilHttp.getParameterMap(request);
    String name = "";
    String firstName = (String)parameters.get("firstName");
    String lastName = (String)parameters.get("lastName");
    if (firstName != null) name += firstName;
    if (lastName != null) {
        if (name != null) name += " ";
        name += lastName;
    }
    message += " " + name + "!";
    message += "<br>" + smallTalk;
    request.setAttribute("_EVENT_MESSAGE_", message);
    return "success";
}
```

Ensure the correct classes have been imported:

```
import org.ofbiz.base.util.UtilProperties;
import java.util.Locale;
```

Shutdown OFBiz, recompile the **Learning Component**, and restart. Fire to webapp learning an http OFBiz request `WelcomeVisitor`. Submit the **First Name** and **Last Name** to see a customized welcome message.

The screenshot shows a web page with a title "The Following Occurred:" followed by a message "Welcome to France! Some Name! The weather is nice." Below this, there is a form with two input fields: "First Name" containing "Some" and "Last Name" containing "Name". A "Submit" button is at the bottom of the form.

First Name	Some
Last Name	Name
Submit	

Note that any changes to the `.properties` files will require an OFBiz restart. Although we can clear the cache in OFBiz, we can't clear the cache in the implementation of `ResourceBundle.getBundle()`.

Parameterizing Messages

Note how we were forced to append the name of the visitor after the message. By parameterizing messages, we can insert the visitor's name into the message.

Change the entry `LearningWelcomeMsg` in the file `${component:learning}\config\LearnMessages.properties` to this:

```
LearningWelcomeMsg=Welcome to France! {0}!
```

In the static method `welcomeVisitor`, delete the line:

```
String message = UtilProperties.getMessage(resource, "welcomeMsg",
                                             locale);
```

and replace:

```
String message += " " + name + "!" ;
```

with:

```
String message = UtilProperties.getMessage(resource,
                                             "LearningWelcomeMsg", new String[] {name}, locale);
```

Once again, shutdown OFBiz, recompile the **Learning Component**, and restart. Fire to webapp learning an http OFBiz request `WelcomeVisitor`, and submit some first and last name. This time, the name is inside of the welcome message, not outside it like before.

The screenshot shows a web page with a title "The Following Occurred:" followed by a message "Welcome to France, Some Name! The weather is nice." Below this, there are two input fields: "First Name" containing "Some" and "Last Name" containing "Name". A blue "Submit" button is at the bottom.

First Name	Some
Last Name	Name
Submit	

Catering for Multiple Languages

It is possible to have OFBiz load different `.properties` files based on the `locale` object. We can have multiple `.properties`, each one catering to a single language.

In the folder `${component:learning}\config`, create a new file `LearnMessages_fr.properties` and enter into it this:

```
LearningApplication=Notre Application D'Etude  
LearningWelcomeMsg= Bienvenue en France, {0}!
```

Restart OFBiz. Let's move on while OFBiz restarts.

The language and country codes are listed online at <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt> and http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html, respectively. Variant codes are non-standard.

Change your browsers language settings to **French** by selecting **Tools | Options** then **Languages** and choosing **French**. Make sure that you move **French** to the top of the list. The process is the same for both Firefox and Internet Explorer.

Fire to webapp learning an http OFBiz request `WelcomeVisitor` and submit a name. We see the French version of the screen appear.



Notice that the `smallTalk` message shows in the English version. OFBiz tries to find the most specific version (`fr` in this case). Failing that, it will fall back on the next general version (in `LearnMessages.properties` in this case).

In fact, much more on the screen has been translated than we entered into our `LearningUiLabels_fr.properties` file. This is because all of the messages in the `CommonUiLabels` have already been translated.

If we had wanted a locale of `fr_CA` (French language in Canada), OFBiz would still have fallen back on `fr`, since that will be the closest and most specific version available.

Note that requesting a more general locale, given only the availability of a more specific one, does not make OFBiz serve up the specific version. For example, asking for `de_DE` will not get `de_DE_no`. In the absence of a general locale and the presence of a specific one, we need to ask for the specific one.

Working with the Database

Although this was covered in detail in Chapter 8, there are a couple of differences between accessing the database in BeanShell and in Java.

Firstly, in Java we must obtain the `GenericDelegator` from the request before we can use it:

```
GenericDelegator delegator =
    (GenericDelegator)request.getAttribute("delegator");
```

Secondly, if we take a look at the class `org.ofbiz.entity.GenericDelegator` we can see all of the methods we were calling in the previous chapter. Virtually all of them throw a `GenericEntityException` which must be handled in our Java code. We must therefore wrap a `try/catch` block around any uses of these methods and handle the error accordingly.

For example:

```
postalAddresses = delegator.findAll("PostalAddress");
```

Would become in our Java code:

```
List postalAddresses = null;
try{
    postalAddresses = delegator.findAll("PostalAddress");
} catch(GenericEntityException e){
    Debug.logError(e, module);
    return "error";
}
```

The `try/catch` block is not needed in BeanShell, because all of the code in the script is ultimately already wrapped in a `try/catch` block and any `Exceptions` thrown are handled by this.

Summary

In this chapter, we have learned a number of vital techniques relevant to programming the meat of the flow in an application. We have learnt more about security and how we create and assign permissions to users and how these permissions can be checked within the Java method. We saw how to invoke Java events from the controller including how to pass in and handle parameters inside the method and how to pass messages and data out of the method. While studying the messages out we took a look at how OFBiz caters for multiple languages.

In the previous chapters we used BeanShell scripts to quickly demonstrate the Entity Engine. In the real world we would port these scripts to compiled Java classes and so we looked at the differences between accessing the delegator's methods in Java versus BeanShell.

In the following chapters we will be covering Service and Minilang environments. Although they fall under the category of Event components, events themselves are large enough to warrant their own chapter.

In this chapter we looked at:

- Creating userLogin records to serve as access identities, much like physical access cards in the real world
- Creating security groups to serve as clearance roles or levels
- Creating security permissions to serve as individual (and indivisible) access rights to individual resources, much like individual secured rooms in the real world
- Classifying security permissions under security groups
- Assigning security groups to userLogins
- Checking access rights in event BeanShell scripts
- Sending feedback from the event to the end-user
- Handling parameters sent to the event codes from the end-user by using `UtilHttp.getParameterMap()`
- Setting up a centralized repository of user-interface words and messages.
- Parameterizing the entries in that repository
- Creating various translations of that repository
- Using the "locale" object to select appropriate language or version of translation for the end-user
- Using Java events to access the database

In the next chapter, we will look at another form of events—Services, and learn more about the Service Engine.

10

The Service Engine

In this chapter, we will be exploring the Service Engine. Services in OFBiz operate in a **Service Oriented Architecture (SOA)**. These services not only have the ability to invoke other services internally, but can also be 'opened up' and invoked by remote applications using, amongst other methods, the widely adopted messaging protocol SOAP.

Besides serving as a platform for interoperability, OFBiz services also offer us additional capability to organize our code. The traditional organizational strategies in object-oriented Java were a great improvement over the procedural paradigm. Wrapping both methods and variables together into objects to form a powerful "behavioral model" for code organization (where object's methods and variables define their behavior). Similarly with OFBiz services we are able to bundle groups of behavior together to form a coherent "service". We can say that OFBiz services, in terms of code or software organization, operate at a higher level than Java object-oriented organizational strategies.

In this chapter, we will be looking at:

- Defining and creating a Java service
- Service parameters
- Special unchecked (unmatched) IN/OUT parameters
- Security-related programming
- Calling services from code (using dispatcher).
- IN/OUT parameter mismatch when calling services
- Sending feedback; standard return codes `success`, `error` and `fail`
- Implementing Service Interfaces
- Synchronous and asynchronous services
- Using the Service Engine tools
- ECAs: Event Condition Actions

Defining a Service

We first need to define a service. Our first service will be named learningFirstService.

In the folder \${component:learning}, create a new folder called servicedef. In that folder, create a new file called services.xml and enter into it this:

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

<services xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "http://www.ofbiz.org/dtds/services.xsd">
  <description>Learning Component Services</description>

  <service name="learningFirstService" engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="learningFirstService">
    <description>Our First Service</description>
    <attribute name="firstName" type="String" mode="IN"
      optional="true"/>
    <attribute name="lastName" type="String" mode="IN"
      optional="true"/>
  </service>
</services>
```

In the file \${component:learning}\ofbiz-component.xml, add after the last <entity-resource> element this:

```
<service-resource type="model" loader="main"
  location="servicedef/services.xml"/>
```

That tells our component learning to look for service definitions in the file \${component:learning}\servicedef\services.xml.



It is important to note that all service definitions are loaded at startup; therefore any changes to any of the service definition files will require a restart!



Creating the Java Code for the Service

In the package org.ofbiz.learning.learning, create a new class called LearningServices with one static method learningFirstService:

```
package org.ofbiz.learning.learning;

import java.util.Map;
```

```
import org.ofbiz.service.DispatchContext;
import org.ofbiz.service.ServiceUtil;

public class LearningServices {
    public static final String module =
        LearningServices.class.getName();

    public static Map learningFirstService(DispatchContext dctx,
        Map context) {
        Map resultMap = ServiceUtil.returnSuccess("You have called on
            service 'learningFirstService' successfully!");
        return resultMap;
    }
}
```

Services must return a map. This map must contain at least one entry. This entry must have the key `responseMessage` (see `org.ofbiz.service.ModelService.RESPONSE_MESSAGE`), having a value of one of the following:

- `success` or `ModelService.RESPOND_SUCCESS`
- `error` or `ModelService.RESPOND_ERROR`
- `fail` or `ModelService.RESPOND_FAIL`

By using `ServiceUtil.returnSuccess()` to construct the minimal return map, we do not need to bother adding the `responseMessage` key and value pair.

Another entry that is often used is that with the key `successMessage` (`ModelService.SUCCESS_MESSAGE`). By doing `ServiceUtil.returnSuccess ("Some message")`, we will get a return map with entry `successMessage` of value "Some message". Again, `ServiceUtil` insulates us from having to learn the convention in key names.

Testing Our First Service

Stop OFBiz, recompile our learning component and restart OFBiz so that the modified `ofbiz-component.xml` and the new `services.xml` can be loaded.

In `${component:learning}\widget\learning\LearningScreens.xml`, insert a new Screen Widget:

```
<screen name="TestFirstService">
    <section>
        <widgets>
            <section>
                <condition><if-empty field-name="formTarget"/></condition>
```

```
<actions>
    <set field="formTarget" value="TestFirstService"/>
    <set field="title" value="Testing Our First Service"/>
</actions>
<widgets/>
</section>
<decorator-screen name="main-decorator"
                  location="${parameters.mainDecoratorLocation}">
    <decorator-section name="body">
        <include-form name="TestingServices"
                      location="component://learning/widget/learning/
LearningForms.xml"/>
    <label text="Full Name: ${parameters.fullName}"/>
    </decorator-section>
</decorator-screen>
</widgets>
</section>
</screen>
```

In the file \${component:learning}\widget\learning\LearningForms.xml, insert a new Form Widget:

```
<form name="TestingServices" type="single" target="${formTarget}">
    <field name="firstName"><text/></field>
    <field name="lastName"><text/></field>
    <field name="planetId"><text/></field>
    <field name="submit"><submit/></field>
</form>
```

Notice how the `formTarget` field is being set in the screen and used in the form. For now don't worry about the **Full Name** label we are setting from the screen. Our service will eventually set that.

In the file \${webapp:learning}\WEB-INF\controller.xml, insert a new request map:

```
<request-map uri="TestFirstService">
    <event type="service" invoke="learningFirstService"/>
    <response name="success" type="view" value="TestFirstService"/>
</request-map>
```

The control servlet currently has no way of knowing how to handle an event of type `service`, so in `controller.xml` we must add a new handler element immediately under the other `<handler>` elements:

```
<handler name="service" type="request"
         class="org.ofbiz.webapp.event.ServiceEventHandler"/>
<handler name="service-multi" type="request"
         class="org.ofbiz.webapp.event.ServiceMultiEventHandler"/>
```

We will cover service-multi services later. Finally add a new view map:

```
<view-map name="TestFirstService" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#TestFirstService"/>
```

Fire to webapp learning an http OFBiz request TestFirstService, and see that we have successfully invoked our first service:

The Following Occurred:
You have called on service 'learningFirstService' successfully!

First Name

Last Name

Planet Id

Submit

Service Parameters

Just like Java methods, OFBiz services can have input and output parameters and just like Java methods, the parameter types must be declared.

Input Parameters (IN)

Our first service is defined with two parameters:

```
<attribute name="firstName" type="String" mode="IN" optional="true"/>
<attribute name="lastName" type="String" mode="IN" optional="true"/>
```

Any parameters sent to the service by the end-user as form parameters, but not in the services list of declared input parameters, will be dropped. Other parameters are converted to a Map by the framework and passed into our static method as the second parameter.

Add a new method handleInputParamaters to our LearningServices class.

```
public static Map handleParameters(DispatchContext dctx, Map
    context) {

    String firstName = (String)context.get("firstName");
    String lastName = (String)context.get("lastName");
    String planetId= (String)context.get("planetId");
```

```
String message = "firstName: " + firstName + "<br/>";  
message = message + "lastName: " + lastName + "<br/>";  
message = message + "planetId: " + planetId;  
  
Map resultMap = ServiceUtil.returnSuccess(message);  
return resultMap;  
}
```

We can now make our service definition invoke this method instead of the `learningFirstService` method by opening our `services.xml` file and replacing:

```
<service name="learningFirstService" engine="java"  
location="org.ofbiz.learning.learning.LearningServices"  
invoke="learningFirstService">
```

with:

```
<service name="learningFirstService" engine="java"  
location="org.ofbiz.learning.learning.LearningServices"  
invoke="handleParameters">
```

Once again shutdown, recompile, and restart OFBiz.

Enter for fields **First Name**, **Last Name**, and **Planet Id** values **Some**, **Name**, and **Earth**, respectively. Submit and notice that only the first two parameters went through to the service. Parameter `planetId` was dropped silently as it was not declared in the service definition.

The Following Occurred:	
firstName:	Some
lastName:	Name
planetId:	null
First Name	<input type="text" value="Some"/>
Last Name	<input type="text" value="Name"/>
Planet Id	<input type="text" value="Earth"/>
Submit	

Modify the service `learningFirstService` in the file `${component:learning}\servicedef\services.xml`, and add below the second parameter a third one like this:

```
<attribute name="planetId" type="String" mode="IN" optional="true"/>
```

Restart OFBiz and submit the same values for the three form fields, and see all three parameters go through to the service.

The Following Occurred:	
firstName:	Some
lastName:	Name
planetId:	Earth
First Name	<input type="text" value="Some"/>
Last Name	<input type="text" value="Name"/>
Planet Id	<input type="text" value="Earth"/>
Submit	

Output Parameters (OUT)

Just like Java methods have return values (although Java methods can have only one typed return value), services can be declared with output parameters. When invoked as events from the controller, parameters will be silently dropped if they are not declared in our service's definition. Add this to our service definition:

```
<attribute name="fullName" type="String" mode="OUT" optional="true"/>
```

And in the method `handleParameters` in `org.ofbiz.learning.learning.LearningServices` replace:

```
Map resultMap = ServiceUtil.returnSuccess(message);
return resultMap;
```

with

```
Map resultMap = ServiceUtil.returnSuccess(message);
resultMap.put("fullName", firstName + " " + lastName);
return resultMap;
```

We have now added the `fullName` parameter to the `resultMap`. To see this in action we need to create a new screen widget in `LearningScreens.xml`:

```
<screen name="TestFirstServiceOutput">
<section>
<actions><set field="formTarget"
value="TestFirstServiceOutput"/></actions>
```

```
<widgets>
    <include-screen name="TestFirstService"/>
</widgets>
</section>
</screen>
```

Add the request-map to the controller.xml file:

```
<request-map uri="TestFirstServiceOutput">
    <event type="service" invoke="learningFirstService"/>
    <response name="success" type="view"
        value="TestFirstServiceOutput"/>
</request-map>
```

and finally the view-map:

```
<view-map name="TestFirstServiceOutput" type="screen"
    page="component://learning/widget/learning/
    LearningScreens.xml#TestFirstServiceOutput"/>
```

Stop OFBiz, rebuild our Learning Component and restart, fire an OFBiz http request TestFirstServiceOutput to webapp learning. Submit your first and last names and planet and notice that now the fullName parameter has been populated.

The Following Occurred:	
firstName:	Some
lastName:	Name
planetId:	Earth
First Name	<input type="text" value="Some"/>
Last Name	<input type="text" value="Name"/>
Planet Id	<input type="text" value="Earth"/>
Submit	
Full Name: Some Name	

Two Way Parameters (INOUT)

A service may change the value of an input parameter and we may need a calling service to be aware of this change. To save us declaring the same parameter twice, with a mode for IN and a mode for OUT, we may use the mode INOUT.

```
<attribute name="fullName" type="String" mode="INOUT"
    optional="true"/>
```

Special Unchecked Parameters

There are a few special cases where IN/OUT parameters can exist even though the service definition does not declare them. They are:

- responseMessage
- errorMessage
- errorMessageList
- successMessage
- successMessageList
- userLogin
- locale

The parameters `responseMessage`, `errorMessage`, `errorMessageList`, `successMessage` and `successMessageList` are necessary placeholders for feedback messages. They must be allowed through all validation checks.

The parameter `userLogin` is often required for authentication and permissions checks.

The parameter `locale` is needed just about everywhere in OFBiz. For locale-specificity in certain operations like retrieving template feedback messages, or like formatting numbers and currency figures.

Optional and Compulsory Parameters

The Service Engine checks the validity of the input and the output to ensure that what is coming into the service and is leaving adheres to the service definition. If the `optional` attribute is set to `false` and an expected parameter is missing, then the validation will fail and the service will return an error. This transaction will now be marked for rollback, meaning any changes to the database made during this transaction will never be committed. This could include any changes made to the database by calling services. For example:

```
<attribute name="fullName" type="String" mode="INOUT"
           optional="false"/>
```

Here the parameter `fullName` must be passed into the service and the service must also add this parameter to the `resultMap` and pass it out or validation will fail and an error will be thrown.

Try changing all of the optional flags on our newly created service to `false`. After a restart we should see:

The Following Errors Occurred:

The following required parameter is missing: [learningFirstService.planetId]

The following required parameter is missing: [learningFirstService.firstName]

The following required parameter is missing: [learningFirstService.lastName]

First Name	<input type="text"/>
Last Name	<input type="text"/>
Planet Id	<input type="text"/>
<input type="button" value="Submit"/>	
Full Name:	

The DispatchContext

We have already seen how parameters are passed into our Java method as a `Map`. Just as the `userLogin` object of type `GenericValue` and the `locale` object of type `Locale` were added as attributes to the request for the Java events, both are now automatically added to this context map when the service is invoked in this way.

The first parameter, the `DispatchContext`, contains the remaining tools we need to access the database, or to invoke other services.

From our Java code we can get access to the following handy objects like this:

```
GenericValue userLogin = (GenericValue) context.get("userLogin");  
  
Locale locale = (Locale) context.get("locale");  
  
GenericDelegator delegator = dctx.getDelegator();  
  
LocalDispatcher dispatcher = dctx.getDelegator();  
  
Security security = dctx.getSecurity();
```

For a full list of objects that are available from the `DispatchContext`, take a look through the code in `org.ofbiz.service.DispatchContext`.

The service engine is in no way reliant on there being `HttpServletRequest` or `HttpServletResponse` objects available. Because of this we are able to invoke services outside of the web environment and they can be invoked remotely or scheduled to run "offline".

Service Security and Access Control

Security-related programming in services is exactly like that in events.

In the class `org.ofbiz.learning.learning.LearningServices`, create a new static method `serviceWithAuth`:

```
public static Map serviceWithAuth(DispatchContext dctx, Map context) {
    Security security = dctx.getSecurity();
    Map resultMap = null;
    if (context.get("userLogin") == null ||
        !security.hasPermission("LEARN_VIEW",
            (GenericValue)context.get("userLogin"))) {
        resultMap = ServiceUtil.returnError("You have no access
            here. You're not welcome!");
    } else {
        resultMap = ServiceUtil.returnSuccess("Welcome! You have
        access!");
    }
    return resultMap;
}
```

Ensure that the correct imports have been added to the class:

```
import java.util.Map;
import org.ofbiz.entity.GenericValue;
import org.ofbiz.security.Security;
```

In the file `${component:learning}\servicedef\services.xml`, add a new service definition:

```
<service name="learningServiceWithAuth" engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="serviceWithAuth">
    <description>Service with some security-related
        codes</description>
</service>
```

In the file `${webapp:learning}\WEB-INF\controller.xml`, add a new request map:

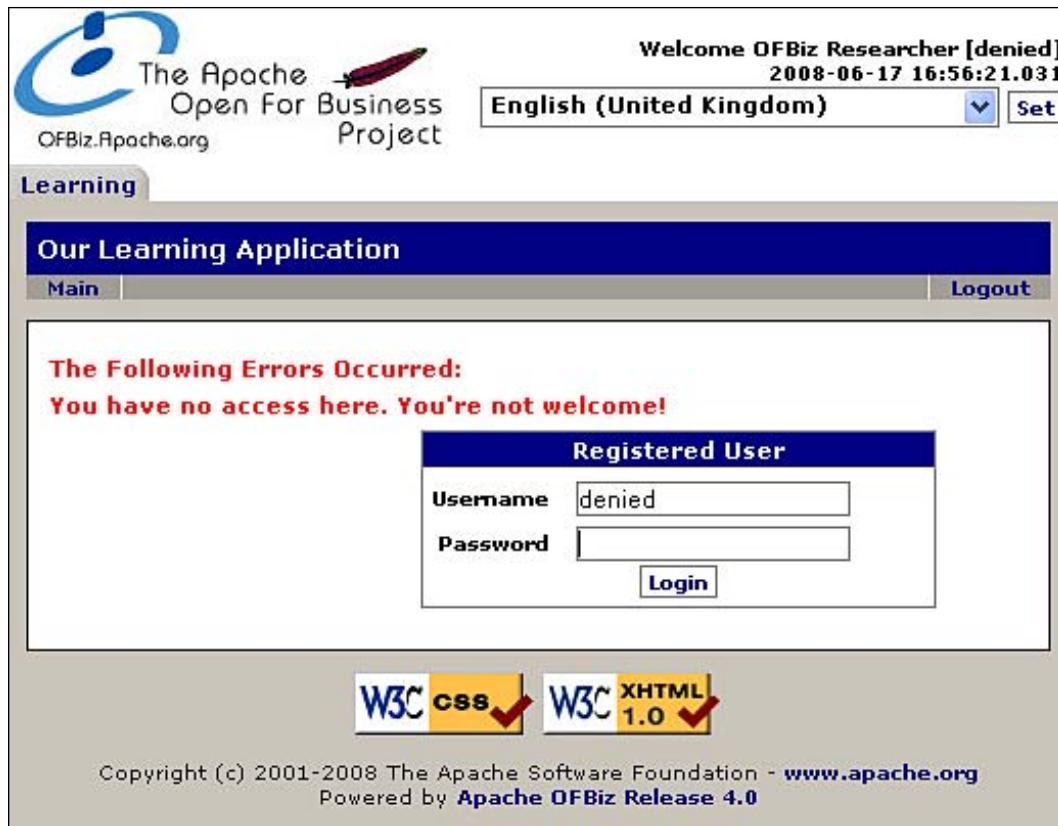
```
<request-map uri="TestServiceWithAuth">
    <security auth="true" https="true"/>
    <event type="service" invoke="learningServiceWithAuth"/>
    <response name="success" type="view" value="SimplestScreen"/>
    <response name="error" type="view" value="login"/>
</request-map>
```

The Service Engine

Rebuild and restart and then fire to webapp learning an http OFBiz request TestServiceWithAuth, login with username **allowed** password **ofbiz**, and see the welcome message displayed:



Logging in with username **denied** password **zibfo** will show an error message. Thanks to the request-map's response element named `error` having a value of `login`, we are returned back to the login screen:



Calling Services from Java Code

So far, we have explored services invoked as events from the controller (example `<event type="service" invoke="learningFirstService"/>`). We now look at calling services explicitly from code.

To invoke services from code, we use the `dispatcher` object, which is an object of type `org.ofbiz.service.ServiceDispatcher`. Since this is obtainable from the `DispatchContext` we can invoke services from other services.

To demonstrate this we are going to create one simple service that calls another.

In our `services.xml` file in `\${component:learning}\servicedef` add two new service definitions:

```
<service name="learningCallingServiceOne" engine="java"
        location="org.ofbiz.learning.learning.LearningServices"
        invoke="callingServiceOne">
```

```
<description>First Service Called From The Controller
</description>
<attribute name="firstName" type="String" mode="IN"
optional="false"/>
<attribute name="lastName" type="String" mode="IN"
optional="false"/>
<attribute name="planetId" type="String" mode="IN"
optional="false"/>
<attribute name="fullName" type="String" mode="OUT"
optional="true"/>
</service>
<service name="learningCallingServiceTwo" engine="java"
location="org.ofbiz.learning.learning.LearningServices"
invoke="callingServiceTwo">
<description>Second Service Called From Service One
</description>
<attribute name="planetId" type="String" mode="IN"
optional="false"/>
</service>
```

In this simple example it is going to be the job of learningCallingServiceOne to prepare the parameter map and pass in the planetId parameter to learningCallingServiceTwo. The second service will determine if the input is EARTH, and return an error if not.

In the class org.ofbiz.learning.learning.LearningEvents, add the static method that is invoked by learningCallingServiceOne:

```
public static Map callingServiceOne(DispatchContext dctx, Map
context) {

    LocalDispatcher dispatcher = dctx.getDispatcher();
    Map resultMap = null;
    String firstName = (String) context.get("firstName");
    String lastName = (String) context.get("lastName");
    String planetId = (String) context.get("planetId");
    GenericValue userLogin = (GenericValue) context.get("userLogin");
    Locale locale = (Locale) context.get("locale");

    Map serviceTwoCtx = UtilMisc.toMap("planetId", planetId,
"userLogin", userLogin, "locale", locale);
    try{
        resultMap = dispatcher.runSync("learningCallingServiceTwo",
serviceTwoCtx);
    }catch(GenericServiceException e){
        Debug.logError(e, module);
    }
}
```

```
        resultMap.put("fullName", firstName + " " + lastName);
        return resultMap;
    }
```

and also the method invoked by learningServiceTwo:

```
public static Map callingServiceTwo(DispatchContext dctx, Map
    context){
    String planetId = (String)context.get("planetId");
    Map resultMap = null;
    if(planetId.equals("EARTH")){
        resultMap = ServiceUtil.returnSuccess("This planet is
            Earth");
    }else{
        resultMap = ServiceUtil.returnError("This planet is NOT
            Earth");
    }
    return resultMap;
}
```

To LearningScreens.xmladd:

```
<screen name="TestCallingServices">
    <section>
        <actions><set field="formTarget" value="TestCallingServices"/></
    actions>
        <widgets>
            <include-screen name="TestFirstService"/>
        </widgets>
    </section>
</screen>
```

Finally add the request-map to the controller.xml file:

```
<request-map uri="TestCallingServices">
    <security auth="false" https="false"/>
    <event type="service" invoke="learningCallingServiceOne"/>
    <response name="success" type="view" value="TestCallingServices"/>
    <response name="error" type="view" value="TestCallingServices"/>
</request-map>
```

and also the view-map:

```
<view-map name="TestCallingServices" type="screen"
    page="component://learning/widget/learning/
    LearningScreens.xml#TestCallingServices"/>
```

Stop, rebuild, and restart, then fire an OFBiz http request TestCallingServices to webapp learning. Do not be alarmed if straight away you see error messages informing us that the required parameters are missing. By sending this request we have effectively called our service with none of our compulsory parameters present.

Enter your name and in the **Planet Id**, enter **EARTH**. You should see:

The Following Occurred:	
This planet is Earth	
First Name	<input type="text" value="Some"/>
Last Name	<input type="text" value="Name"/>
Planet Id	<input type="text" value="EARTH"/>
Submit	
Full Name: Some Name	

Try entering **MARS** as the **Planet Id**.

The Following Errors Occurred:	
This planet is NOT Earth	
First Name	<input type="text" value="Some"/>
Last Name	<input type="text" value="Name"/>
Planet Id	<input type="text" value="MARS"/>
Submit	
Full Name: Some Name	

Notice how in the Java code for the static method `callingServiceOne` the line

```
resultMap = dispatcher.runSync("learningCallingServiceTwo",  
                           serviceTwoCtx);
```

is wrapped in a try/catch block. Similar to how the methods on the `GenericDelegator` object that accessed the database threw a `GenericEntityException`, methods on our `dispatcher` object throw a `GenericServiceException` which must be handled.

There are three main ways of invoking a service:

- `runSync` – which runs a service synchronously and returns the result as a map.

- `runSyncIgnore` – which runs a service synchronously and ignores the result. Nothing is passed back.
- `runAsync` – which runs a service asynchronously. Again, nothing is passed back.

The difference between synchronously and asynchronously run services is discussed in more detail in the section called *Synchronous and Asynchronous Services*.

Implementing Interfaces

Open up the `services.xml` file in `${component:learning}\servicedef` and take a look at the service definitions for both `learningFirstService` and `learningCallingServiceOne`.

Do you notice that the `<attribute>` elements (parameters) are the same? To cut down on the duplication of XML code, services with similar parameters can implement an interface.

As the first service element in this file enter the following:

```
<service name="learningInterface" engine="interface">
    <description>Interface to describe base parameters for Learning
        Services</description>
    <attribute name="firstName" type="String" mode="IN"
        optional="false"/>
    <attribute name="lastName" type="String" mode="IN"
        optional="false"/>
    <attribute name="planetId" type="String" mode="IN"
        optional="false"/>
    <attribute name="fullName" type="String" mode="OUT"
        optional="true"/>
</service>
```

Notice that the `engine` attribute is set to `interface`.

Replace all of the `<attribute>` elements in the `learningFirstService` and `learningCallingServiceOne` service definitions with:

```
<implements service="learningInterface"/>
```

So the service definition for `learningServiceOne` becomes:

```
<service name="learningCallingServiceOne" engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="callingServiceOne">
```

```
<description>First Service Called From The Controller
</description>
<implements service="learningInterface"/>
</service>
```

Restart OFBiz and then fire an OFBiz http request TestCallingServices to webapp learning. Nothing should have changed—the services should run exactly as before, however our code is now somewhat tidier.

Overriding Implemented Attributes

It may be the case that the interface specifies an attribute as `optional="false"`, however, our service does not need this parameter. We can simply override the interface and add the `<attribute>` element with whatever settings we wish.

For example, if we wish to make the `planetId` optional in the above example, the `<implements>` element could remain, but a new `<attribute>` element would be added like this:

```
<service name="learningCallingServiceOne" engine="java"
         location="org.ofbiz.learning.learning.LearningServices"
         invoke="callingServiceOne">
    <description>First Service Called From The Controller
    </description>
    <implements service="learningInterface"/>
    <attribute name="planetId" type="String" mode="IN"
              optional="false"/>
</service>
```

Synchronous and Asynchronous Services.

The service engine allows us to invoke services synchronously or asynchronously. A synchronous service will be invoked in the same thread, and the thread will "wait" for the invoked service to complete before continuing. The calling service can obtain information from the synchronously run service, meaning its OUT parameters are accessible.

Asynchronous services run in a separate thread and the current thread will continue without waiting. The invoked service will effectively start to run in parallel to the service or event from which it was called. The current thread can therefore gain no information from a service that is run asynchronously. An error that occurs in an asynchronous service will not cause a failure or error in the service or event from which it is called.

A good example of an asynchronously called service is the `sendOrderConfirmation` service that creates and sends an order confirmation email. Once a customer has placed an order, there is no need to wait while the mail service is called and the mail sent. The mail server may be down, or busy, which may result in an error that would otherwise stop our customer from placing the order. It is much more preferable to allow the customer to continue to the **Order Confirmation** page and have our business receive the valuable order. By calling this service asynchronously, there is no delay to the customer in the checkout process, and while we log and fix any errors with the mail server, we still take the order.

Behind the scenes, an asynchronous service is actually added to the Job Scheduler. It is the Job Scheduler's task to invoke services that are waiting in the queue.

Using the Job Scheduler

Asynchronous services are added to the Job Scheduler automatically. However, we can see which services are waiting to run and which have already been invoked through the **Webtools** console. We can even schedule services to run once only or recur as often as we like.

Open up the **Webtools** console at `https://localhost:8443/webtools/control/main` and take a look under the **Service Engine Tools** heading. Select **Job List** to view a full list of jobs. Jobs without a **Start Date/Time** have not started yet. Those with an **End Date/Time** have completed. The **Run Time** is the time they are scheduled to run. All of the outstanding jobs in this list were added to the **JobSandbox Entity** when the initial seed data load was performed, along with the **RecurrenceRule** (also an Entity) information specifying how often they should be run. They are all maintenance jobs that are performed "offline".

The **Pool** these jobs are run from by default is set to `pool1`. In an architecture where there may be multiple OFBiz instances connecting to the same database, this can be important. One OFBiz instance can be dedicated to performing certain jobs, and even though job schedulers may be running on each instance, this setting can be changed so we know only one of our instances will run this job.

The Service Engine settings can be configured in `framework\service\config\serviceengine.xml`. By changing both the `send-to-pool` attribute and the `name` attribute on the `<run-from-pool>` element, we can ensure that only jobs created on an OFBiz instance are run by this OFBiz instance.

Click on the **Schedule Job** button and in the Service field enter learningCallingServiceOne, leave the **Pool** as pool and enter today's date/time by selecting the calendar icon and clicking on today's date. We will need to add 5 minutes onto this once it appears in the box. In the below example the Date appeared as **2008-06-18 14:16:24.265**. This job is only going to be scheduled to run once, although we could specify any recurrence information we wish.

Service List	Job List	Thread List	Schedule Job	Run Service
Step 1: Service And Recurrence Information				
Job	Test Scheduled Job			
Service	learningCallingServiceOne			
Pool	pool			
Start Date/Time	2008-06-18 14:16:24.265			
End Date/Time	<input type="text"/>			
Frequency	None			
Interval	<input type="text"/> <small>for use with frequency</small>			
Count	1 <small>number of time the job will run; use -1 for no limit i.e. forever</small>			
Max Retry	<input type="text"/> <small>number of time the job will retry on error; use -1 for no limit or leave empty for service default</small>			
Submit				

Select **Submit** and notice that scheduler is already aware of the parameters that can (or must, in this case) be entered. This information has been taken from the service definition in our services.xml file.

Service List	Job List	Thread List	Schedule Job	Run Service
Step 2: Service Parameters				
firstName (String)	<input type="text"/> Some		(required)	
lastName (String)	<input type="text"/> Name		(required)	
planetId (String)	<input type="text"/> EARTH		(required)	
Submit				

Press **Submit** to schedule the job and find the entry in the list. This list is ordered by **Run Time** so it may not be the first. Recurring maintenance jobs are imported in the seed data and are scheduled to run overnight. These will more than likely be above the job we have just scheduled since their run-time is further in the future. The entered parameters are converted to a map and then serialized to the database. They are then fed to the service at run time.

Test Scheduled Job	10163 pool 2008-06-18 14:16:24.265	learningCallingServiceOne	Cancel Job
--------------------	---------------------------------------	---------------------------	----------------------------

Quickly Running a Service

Using the **Webtools** console it is also possible to run a service synchronously. This is quicker than going through the scheduler should you need to test a service or debug through a service. Select the **Run Service** button from the menu and enter the same service name, **submit** then enter the same parameters again. This time the service is run straight away and the OUT parameters and messages are passed back to the screen:

The Following Occurred: Service has been scheduled		
parameter	value	save value?
responseMessage	success	<input type="checkbox"/>
successMessage	This planet is Earth	<input type="checkbox"/>
fullName	Some Name	<input type="checkbox"/>
	Clear previous params? <input type="checkbox"/>	submit

Naming a Service and the Service Reference

Service names must be unique throughout the entire application. Because we do not need to specify a location when we invoke a service, if service names were duplicated we can not guarantee that the service we want to invoke is the one that is actually invoked. OFBiz comes complete with a full service reference, which is in fact a dictionary of services that we can use to check if a service exists with the name we are about to choose, or even if there is a service already written that we are about to duplicate.

From <https://localhost:8443/webtools/control/main> select the **Service Reference** and select "1" for learning. Here we can see all of our learning services, what engine they use and what method they invoke. By selecting the service `learningCallingServiceOne`, we can obtain complete information about this service as was defined in the service definition file `services.xml`. It even includes information about the parameters that are passed in and out automatically.

Careful selection of intuitive service names and use of the description tags in the service definition files are good practice since this allows other developers to reuse services that already exists, rather than duplicate work unnecessarily.

Service: learningCallingServiceOne							Schedule	List All
Service Name:	learningCallingServiceOne		Engine Name:	java				
Description:	First Service Called From The Controller		Invoke:	callingServiceOne				
Exportable:	False		Location:	org.ofbiz.learning.learning.LearningServices				
			Default Entity Name:	NA				
			Require new transaction:	False				
			Use transaction:	True				
			Max retries:	-1				
Security Groups								
NA								
Implemented Services								
NA								
In parameters								
Parameter Name	Optional	Type	Mode	Is set internally	Entity Name	Field Name		
firstName	False	String	IN	False				
lastName	False	String	IN	False				
planetid	False	String	IN	False				
userLogin	True	org.ofbiz.entity.GenericValue	INOUT	True				
locale	True	java.util.Locale	INOUT	True				
Out parameters								
Parameter Name	Optional	Type	Mode	Is set internally	Entity Name	Field Name		
fullName	True	String	OUT	False				
responseMessage	True	String	OUT	True				
errormessage	True	String	OUT	True				
errorMessageList	True	java.util.List	OUT	True				
successMessage	True	String	OUT	True				
successMessageList	True	java.util.List	OUT	True				
userLogin	True	org.ofbiz.entity.GenericValue	INOUT	True				
locale	True	java.util.Locale	INOUT	True				

Event Condition Actions (ECAs)

ECA refers to the structure of rules of a process. The Event is the trigger or the reason why the rule is being invoked. The condition is a check to see if we should continue and invoke the action, and the action is the final resulting change or modification. A real life example of an ECA could be "If you are leaving the house, check to see if it is raining. If so, fetch an umbrella". In this case the event is "leaving the house". The condition is "if it is raining" and the action is "fetch an umbrella".

There are two types of ECA rules in OFBiz: **Service Event Condition Actions (SECAs)** and **Entity Event Condition Actions (EECAs)**.

Service Event Condition Actions (SECAs)

For SECAs the trigger (Event) is a service being invoked. A condition could be if a parameter equalled something (conditions are optional), and the action is to invoke another service.

SECAs are defined in the same directory as service definitions (`servicedef`). Inside files named `secas.xml`

Take a look at the existing SECAs in `applications\order\servicedef\secas.xml` and we can see a simple ECA:

```
<eca service="changeOrderStatus" event="commit"
      run-on-error="false">
    <condition field-name="statusId" operator="equals"
               value="ORDER_CANCELLED"/>
    <action service="releaseOrderPayments" mode="sync"/>
</eca>
```

When the `changeOrderStatus` transaction is just about to be committed, a lookup is performed by the framework to see if there are any ECAs for this event. If there are, and the parameter `statusId` is `ORDER_CANCELLED` then the `releaseOrderPayments` service is run synchronously.

Most commonly, SECAs are triggered on `commit` or `return`; however, it is possible for the event to be in any of the following stages in the service's lifecycle:

- `auth`—Before Authentication
- `in-validate`—Before IN parameter validation
- `out-validate`—Before OUT parameter validation
- `invoke`—Before service invocation
- `commit`—Just before the transaction is committed
- `return`—Before the service returns
- `global-commit`
- `global-rollback`

The variables `global-commit` and `global-rollback` are a little bit different. If the service is part of a transaction, they will only run after a rollback or between the two phases (JTA implementation) of a commit.

There are also two specific attributes whose values are false by default:

- `run-on-failure`
- `run-on-error`

You can set them to true if you want the SECA to run in spite of a failure or error. A failure is the same thing as an error, except it doesn't represent a case where a rollback is required.

It should be noted that parameters passed into the trigger service are available, if need be, to the action service. The trigger services OUT parameters are also available to the action service.

Before using SECAs in a component, the component must be informed of the location of the ECA service-resources:

```
<service-resource type="eca" loader="main"  
                  location="servicedef/secas.xml"/>
```

This line must be added under the existing `<service-resource>` elements in the component's `ofbiz-component.xml` file.

Entity Event Condition Actions (EECAs)

For EECAs, the event is an operation on an entity and the action is a service being invoked.

EECAs are defined in the same directory as entity definitions (`entitydef`): inside files named `eecas.xml`.

They are used when it may not necessarily be a service that has initiated an operation on the entity, or you may wish that no matter what service operates on this entity, a certain course of action to be taken.

Open the `eecas.xml` file in the `applications\product\entitydef` directory and take a look at the first `<eca>` element:

```
<eca entity="Product" operation="create-store" event="return">  
    <condition field-name="autoCreateKeywords"  
               operator="not-equals" value="N"/>  
    <action service="indexProductKeywords" mode="sync"  
           value-attr="productInstance"/>  
</eca>
```

This ECA ensures that once any creation or update operation on a Product record has been committed, so long as the `autoCreateKeywords` field of this record is not N, then the `indexProductKeywords` service will be automatically invoked synchronously.

The operation can be any of the following self-explanatory operations:

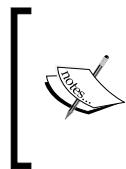
- `create`
- `store`
- `remove`
- `find`
- `create-store` (create or store/update)
- `create-remove`
- `store-remove`
- `create-store-remove`
- `any`

The `return` event is by far the most commonly used event in an EECA. But there are also `validate`, `run`, `cache-check`, `cache-put`, and `cache-clear` events. There is also the `run-on-error` attribute.

Before using EECAs in a component, the component must be informed of the location of the eca entity-resource:

```
<entity-resource type="eca" loader="main"  
location="entitydef/eecas.xml"/>
```

must be added under the existing `<entity-resource>` elements in the component's `ofbiz-component.xml` file.



ECAs can often catch people out! Since there is no apparent flow from the trigger to the service in the code they can be difficult to debug. When debugging always keep an eye on the logs. When an ECA is triggered, an entry is placed into the logs to inform us of the trigger and the action.

Summary

This brings us to the end of our investigation into the OFBiz Service Engine. We have discovered how useful the Service Oriented Architecture in OFBiz can be and we have learnt how the use of some of the built in Service Engine tools, like the Service Reference, can help us when we are creating new services.

In this chapter we have looked at:

- Defining and creating services
- Service parameters
- Special unchecked (unmatched) IN/OUT parameters
- Security-related programming
- Calling services from code (using `dispatcher`).
- IN/OUT parameter mismatch when calling services
- Sending feedback; standard return codes `success`, `error` and `fail`.
- Implementing Service Interfaces
- Synchronous and asynchronous services
- Using the Service Engine tools
- ECAs: Event Condition Actions

The Service Engine is highly developed with respect to permissions and access control. In the next chapter we will be studying OFBiz Permissions and the Service Engine.

11

Permissions and the Service Engine

The Service Engine allows us to determine who can and who can't invoke services based on permissions given to user logins. In fact, the Service Engine allows us to go beyond simple permissions attached to the user login. As we will see in this chapter, it is possible to easily create our own checks for any criteria to determine whether the logged-in user is allowed to invoke a service.

In this chapter we will be exploring:

- Simple permissions, entity permission, and role checks
- Combining and nesting those permissions and checks
- Permission service (really complex permissions)

Simple Permissions

Each service can be declared with required permissions. The `dispatcher` object (`ServiceDispatcher`) will check the specific permissions before invoking the service.

We specify required permissions with the element `<required-permissions>`.

To the file `${component:learning}\servicedef\services.xml` add a new service definition called `learningCallingServiceOneWithPermission`. This will invoke the same method as the previous definition for `learningCallingServiceOne`:

```
<service name="learningCallingServiceOneWithPermission"
    engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="callingServiceOne">
```

```
<description>First Service Called From The  
Controller</description>  
<required-permissions join-type="OR">  
    <check-permission permission="LEARN_VIEW"/>  
</required-permissions>  
    <implements service="learningInterface"/>  
</service>
```

The `<required-permissions>` element has only one attribute, `join-type`, and it is mandatory. The values for `join-type` are AND or OR, and they specify how multiple permission checks are joined together logically.

The `<required-permissions>` element can contain any number of `<check-permission>` and `<check-role-member>` elements (discussed later).

Simple permissions are checked with `<check-permission>` elements.

To the learning component's `controller.xml` file add the following two request-maps:

```
<request-map uri="TestPermissions">  
    <security auth="true" https="true"/>  
    <response name="success" type="view"  
        value="TestCallingServicesWithPermission"/>  
    <response name="error" type="view" value="login"/>  
</request-map>  
<request-map uri="TestCallingServicesWithPermission">  
    <security auth="true" https="true"/>  
    <event type="service"  
        invoke="learningCallingServiceOneWithPermission"/>  
    <response name="success" type="view"  
        value="TestCallingServicesWithPermission"/>  
    <response name="error" type="view"  
        value="TestCallingServicesWithPermission"/>  
</request-map>
```

And the view-map:

```
<view-map name="TestCallingServicesWithPermission" type="screen"  
    page="component://learning/widget/learning/  
        LearningScreens.xml#TestCallingServicesWithPermission"/>
```

Finally, we need to add a simple screen widget `TestCallingServicesWithPermission`:

```
<screen name="TestCallingServicesWithPermission">  
    <section>  
        <actions><set field="formTarget"  
            value="TestCallingServicesWithPermission"/>
```

```

</actions>
<widgets>
    <include-screen name="TestFirstService"/>
</widgets>
</section>
</screen>

```

Restart OFBiz.

Fire an http OFBiz request TestPermission to webapp learning, and login as **allowed** (password **ofbiz**). Notice how this time we added two request-maps to the controller. The first one gives us access to the screen without invoking the service. Had we accessed the second request-map directly, the service would have been invoked, and since we have not passed the service the required parameters, the first thing we would have seen is an error. This technique is common in OFBiz.

Enter the fields accordingly. We see our call to service learningCallingServiceOneWithPermission was successful, and the service composed for us a full name from our input parameters:

The screenshot shows a web page with a form. At the top, it says "The Following Occurred:" followed by "This planet is Earth". Below this, there are three input fields: "First Name" with value "Some", "Last Name" with value "Name", and "Planet Id" with value "EARTH". There is also a "Submit" button. At the bottom of the form, it displays "Full Name: Some Name".

Follow the same procedure with the user login **denied**, and see that our call to service learningCallingServiceOneWithPermission throws up an exception. The exception means that we do not have the required permission to access that service.

The screenshot shows a web page with a red error message box. It says "The Following Errors Occurred:" followed by "You do not have permission to invoke the service [learningCallingServiceOneWithPermission]". Below this, there are three input fields: "First Name" with value "Some", "Last Name" with value "Name", and "Planet Id" with value "EARTH". There is also a "Submit" button. At the bottom of the form, it displays "Full Name:".

Two-Part Permissions and Special "ADMIN" Permissions

Two-part permissions are representative of permission strings of the form <main-permission>_<action>. For example, the permission LEARN_VIEW will have a permission of LEARN and an action of VIEW.

In the file \${component:learning}\servicedef\services.xml, modify the service declaration learningCallingServiceOneWithPermission and change its <check-permission> element to this:

```
<check-permission permission="LEARN" action="VIEW"/>
```

Restart OFBiz, and follow the same procedure as with the previous section.

We'll see that this:

```
<check-permission permission="LEARN" action="VIEW"/>
```

is really the same as:

```
<check-permission permission="LEARN_VIEW"/>
```

Whenever the action attribute is present in the <check-permission>, it becomes a two-part permission, not just a simple one.

A two-part permission also has a special handling for general access privileges, like those for a "super user". A general access privilege, or super permission, is security permission that grants access to all parts of an application. Such an umbrella privilege will mean we just need to grant that single super permission to the "super user", rather than having to grant possibly thousands of security permissions in order to cover all bases.

In the **Entity Data Maintenance** screen for the entity SecurityPermission, <https://localhost:8443/webtools/control/FindGeneric?entityName=SecurityPermission>, create a new permission LEARN_ADMIN, with the field **description** given a value of **Super access for learning screens**.

Fire an https OFBiz request to webapp partymgr (<https://localhost:8443/partymgr>), login as **admin**, select **Security** then **New Security Group** and create a new security group LEARNALL with a **description** of **Super access for learning**.

Assign the security group LEARNALL to the user login **denied**.

Perform the same experiment we did for simple permissions above, and see that the user login **denied** now has access to the service `learningCallingServiceOneWithPermission`. The user login **denied** only has security permission `LEARN_ADMIN` (via security group `LEARNALL`).

Now, change the entity permission in `services.xml` back to the simple permission above (attribute `permission` of `LEARN_VIEW`, no `action` attribute). The user login **denied** will now have no access to the `learningCallingServiceOneWithPermission`, even though it has the catch-all `_ADMIN` security permission. Therefore, it is more common to use two-part permissions than simple permissions, especially when we need to use the special `_ADMIN` catch-all security permissions.

Role Checks

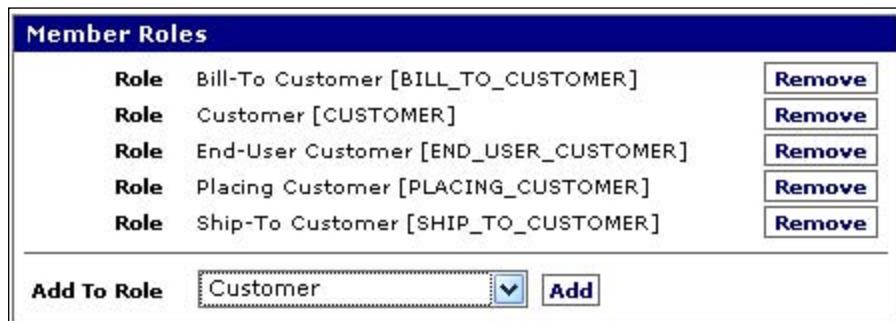
Besides security permission checks, we can also do role checks. In typical systems, roles would be equivalent to the OFBiz security groups; a means to group security permissions together in a "role-based" style of security system. In OFBiz, roles are like another category of "permissions"; roles could have been called "role permissions", since they function quite like security permissions. However, roles are more than mere permissions. In OFBiz, they are used to determine what kinds of relationships are possible between parties (example, a party with the role "supplier" can be linked to parties with the role "customer").

In practice, for the purposes of security and access control, we can consider roles as broad-based permissions, and as such security groups will be fine-grained access control, and security permissions will be the finest units of access control. If we consider this to be similar to the security of a building, having the role "Employee" may give us access to the front door of the building. Having a security group of `LEARNING` may give us access to a floor and having a permission of `LEARNING_VIEW` would give us access to a room.

Fire an http OFBiz request to webapp partymgr, and at the **Find Party** screen, look for the user login **allowed**. We will be brought to the party profile (also entity) Person **OFBiz Researcher** (first name **OFBiz**, last name **Researcher**). At the top right of the profile screen, click the **Role(s)** tab (second tab from left, on the right of **Profile**).



That'll bring us to the **Member Roles** screen for the Person **OFBiz Researcher**. In the following experiments, we'll be using the role **CUSTOMER**. If the Person (with user logins **allowed** and **denied**) you will be experimenting with doesn't have that role, you can add it manually. We will be also be using role **ADMIN** in negative experiments, so make sure the Person doesn't have this role.



In the file \${component:learning}\servicedef\services.xml, edit the <required-permissions> element to replace this:

```
<check-permission permission="LEARN" action="VIEW" />
```

with this:

```
<check-role-member role-type="CUSTOMER" />
```

Before we move on, we need to fix a bug in OFBiz's execution of `<check-role-member>` elements. At one point in time the Entity `PartyRole` had `fromDate` and `thruDate` fields, and the roles are still filtered using the Post Query Filtering method described in Chapter 8. However, since the date fields have now been moved to a different Entity, filtering by date is no longer possible. If this bug hasn't already been fixed by the time of publication, go to the folder `${component:service}\src\org\ofbiz\service` and edit the file `ModelPermission.java`. Comment out line 103 like this:

```
//partyRoles = EntityUtil.filterByDate(partyRoles);
```

Stop OFBiz and recompile the whole application by double-clicking the OFBiz Main Build target in the **Ant Window** of Eclipse.

Fire to webapp learning an http OFBiz request `TestPermissions`, and login as **allowed** or **denied**. Both user logins are tied to the same `Person` (actually the `Party` entity) which has the role `CUSTOMER`. Therefore, both user logins will be able to invoke the service `learningCallingServiceOneWithPermission` successfully.

In the file `${component:learning}\servicedef\services.xml`, edit the `<required-permissions>` element to replace this:

```
<check-role-member role-type="CUSTOMER"/>
```

with this:

```
<check-role-member role-type="ADMIN"/>
```

Restart OFBiz. Since the Person involved does not have the role `ADMIN`, neither user logins can now invoke `learningCallingServiceOneWithPermission`.

Combining Multiple Checks

Using the attribute `join-type` of `<required-permissions>` elements, it is possible to combine multiple checks.

In the file `${component:learning}\servicedef\services.xml`, edit the `<required-permissions>` element to replace this:

```
<check-role-member role-type="ADMIN"/>
```

with this:

```
<check-role-member role-type="CUSTOMER"/>
<check-role-member role-type="ADMIN"/>
```

Make sure the `join-type` attribute of the `<required-permissions>` element is `OR`. It should already be so; we haven't changed it since we created it.

Obviously, either user login can invoke `learningCallingServiceOneWithPermission`. Although the Person involved does not have the role `ADMIN`, it does have the role `CUSTOMER`.

In the file `${component:learning}\servicedef\services.xml`, edit the `<required-permissions>` and change its `join-type` to `AND`. Also, replace this:

```
<check-role-member role-type="CUSTOMER"/>
<check-role-member role-type="ADMIN"/>
```

with this:

```
<check-permission permission="LEARN_VIEW"/>
<check-role-member role-type="CUSTOMER"/>
```

This time, only the user login **allowed** can get through to the service `learningCallingServiceOneWithPermission`; it has security permission `LEARN_VIEW` and its Person has role `CUSTOMER`. User login **denied** only qualifies where its Person has role `CUSTOMER`, but the user login doesn't have permission `LEARN_VIEW`. Therefore, user login **denied** won't get through.

Nested Checks

We can have multiple `<required-permissions>` for a service declaration. That means we can achieve a maximum of two layers of nested checks. Multiple `<required-permissions>` are combined together by `AND`.

In the file `${component:learning}\servicedef\services.xml`, edit the service declaration `learningCallingServiceOneWithPermission` to change its `<required-permissions>` element to this:

```
<required-permissions join-type="OR">
  <check-permission permission="LEARN_VIEW"/>
</required-permissions>
<required-permissions join-type="OR">
  <check-role-member role-type="CUSTOMER"/>
</required-permissions>
```

Fire to webapp learning an http OFBiz request `TestPermissions`, and login as **allowed** or **denied**. Only the user login **allowed** can get through to the service `learningCallingServiceOneWithPermission`; it has the security permission `LEARN_VIEW` and its Person has the role `CUSTOMER`. The user login **denied** does not have the security permission `LEARN_VIEW` (only `LEARN_ADMIN`). As can be seen, the above `<required-permissions>` is equivalent to:

```
<required-permissions join-type="AND">
  <check-permission permission="LEARN_VIEW"/>
  <check-role-member role-type="CUSTOMER"/>
</required-permissions>
```

Nested checks can at most be two layers/levels deep. The `<required-permissions>` element cannot contain `<required-permissions>` elements. Nor can its children (`<check-permission>` and `<check-role-member>`) contain elements of their own type.

Complex Permissions

More complex permissions are possible with the `<permission-service>` element. A complex permission programmed with a service can have limitless possibilities, as complex as we can manage to code the service.

Setting-Up Our Playground

Before we play with `<permission-service>` elements, we need to set up our playground. Complex permissions can quickly get baffling if we don't set up a clean and structured playground designed for controlled experiments.

In this example we will create a form for end-users to write reviews about the planets they live on. These reviews will be stored in the entity `PlanetReview`. The above two services handle the creation and modification of `PlanetReview` records.

For permissions, we will restrict end-users to writing reviews only about the planets on which they have a postal address. We won't allow them to write about planets they don't live on.

Additionally, we will allow only the original user login to update the reviews; each review will be recorded with a link to the user login that created it. This will ensure that only the original creator of a review can modify the review. Of course, we know by now that multiple user logins can belong to the same `Person`; in our experiments, user logins allowed and denied actually do belong to the same `Person`. However, rather than bothering ourselves with creating another `Person`, we will be treating different logins as though they are different people.

Creating the Request and View Maps

In the file \${webapp:learning}\WEB-INF\controller.xml, add four new request maps:

```
<!-- Planet Review Request Maps -->
<request-map uri="PlanetReview">
    <security auth="true" https="true"/>
    <response name="success" type="view" value="PlanetReview"/>
</request-map>
<request-map uri="createPlanetReview">
    <security auth="true" https="true"/>
    <event type="service" invoke="learningCreatePlanetReview"/>
    <response name="success" type="view" value="PlanetReview"/>
</request-map>
<request-map uri="updatePlanetReview">
    <security auth="true" https="true"/>
    <event type="service" invoke="learningUpdatePlanetReview"/>
    <response name="success" type="view" value="PlanetReview"/>
</request-map>
<request-map uri="ListPlanetReviews">
    <security auth="true" https="true"/>
    <response name="success" type="view" value="ListPlanetReviews"/>
</request-map>
<!-- End Planet Review Request Maps -->
```

and two new view maps:

```
<view-map name="PlanetReview" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#PlanetReview"/>
<view-map name="ListPlanetReviews" type="screen"
    page="component://learning/widget/learning/
        LearningScreens.xml#ListPlanetReviews"/>
```

Creating the Screen and Form Widgets

In the file \${component:learning}\widget\learning\LearningScreens.xml, add two new screen widgets:

```
<screen name="PlanetReview">
    <section>
        <!-- The entity-one element will try to find field reviewId from
            ${parameters.reviewId}. -->
        <actions><entity-one entity-name="PlanetReview"
            value-name="planetReview"/></actions>
```

```

<widgets>
    <decorator-screen name="main-decorator"
                      location="${parameters.mainDecoratorLocation}">
        <decorator-section name="body">
            <include-form name="PlanetReview"
                          location="component://learning/widget/learning/
                           LearningForms.xml"/>
        </decorator-section>
    </decorator-screen>
</widgets>
</section>
</screen>

<screen name="ListPlanetReviews">
    <section>
        <actions><entity-condition entity-name="PlanetReview"
                                    list-name="planetReviews"/></actions>
        <widgets>
            <decorator-screen name="main-decorator"
                              location="${parameters.mainDecoratorLocation}">
                <decorator-section name="body">
                    <include-form name="PlanetReviews"
                                  location="component://learning/widget/learning/
                                   LearningForms.xml"/>
                </decorator-section>
            </decorator-screen>
        </widgets>
    </section>
</screen>

```

In the file \${component:learning}\widget\learning\LearningForms.xml, add two new Form Widgets:

```

<form name="PlanetReview" type="single" target="createPlanetReview"
      default-map-name="planetReview">
    <alt-target target="updatePlanetReview"
               use-when="planetReview!=null"/>
    <field name="reviewId"
          use-when="planetReview!=null"><display/></field>
    <field name="createdBy" use-when="planetReview!=null">
        <display description="${planetReview.userLoginId}" />
    </field>
    <field name="planetId">
        <drop-down current="selected">
            <entity-options entity-name="Planet"
                           description="${planetName}" />

```

```
<entity-order-by field-name="planetName"/>
</entity-options>
</drop-down>
</field>
<field name="review"><textarea/></field>
<field name="submit"><submit/></field>
</form>

<form name="PlanetReviews" type="list" target="PlanetReview"
      list-name="planetReviews">
    <field name="reviewId"><display/></field>
    <field name="createdBy">
        <display description="\${userLoginId}"/>
    </field>
    <field name="planetId"><display/></field>
    <field name="review"><display/></field>
    <field name="update"><submit/></field>
</form>
```

Creating the Entity PlanetReview

In the file \${component:learning}\entitydef\entitymodel.xml, add a new Entity:

```
<entity entity-name="PlanetReview" package-name="org.ofbiz.learning">
    <field name="reviewId" type="id-ne"/>
    <field name="userLoginId" type="id-vlong-ne"/>
    <field name="planetId" type="id-ne"/>
    <field name="review" type="very-long"/>
    <prim-key field="reviewId"/>
    <relation type="one" fk-name="PREVIEW_PLANET"
              rel-entity-name="Planet">
        <key-map field-name="planetId"/>
    </relation>
    <relation type="one" fk-name="PREVIEW_ULOGIN"
              rel-entity-name="UserLogin">
        <key-map field-name="userLoginId"/>
    </relation>
</entity>
```

In the file \${component:learning}\entitydef\entitygroup.xml, assign the new entity to the entity group "org.ofbiz" like this:

```
<entity-group group="org.ofbiz" entity="PlanetReview"/>
```

Defining Services that Require Complex Permission

We now define our first services that use complex permissions.

In file \${component:learning}\servicedef\services.xml, add two new service definitions:

```
<service name="learningCreatePlanetReview" engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="createPlanetReview">
    <permission-service
        service-name="learningCheckCreatePlanetReviewPermission"/>
    <attribute name="planetId" type="String" mode="IN"/>
    <attribute name="review" type="String" mode="IN"/>
    <attribute name="reviewId" type="String" mode="OUT"/>
</service>

<service name="learningUpdatePlanetReview" engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="updatePlanetReview ">
    <permission-service
        service-name="learn.checkUpdatePlanetReviewPermission"/>
    <attribute name="reviewId" type="String" mode="IN"/>
    <attribute name="planetId" type="String" mode="IN"/>
    <attribute name="review" type="String" mode="IN"/>
    <attribute name="reviewId" type="String" mode="OUT"/>
</service>
```

Implementing Services that Require Complex Permission

Implementing services that require complex permissions is just like doing so for services without. That's the beauty of using permissions; the permissions logic is cleanly segregated from core business logic.

We now implement the two create and update services using Java.

In the class org.ofbiz.learning.learning.LearningServices, create a new static method `createPlanetReview`:

```
public static Map createPlanetReview(DispatchContext dctx, Map
context) {
    GenericDelegator delegator = dctx.getDelegator();
    GenericValue userLogin =
        (GenericValue) context.get("userLogin");
```

```
if (userLogin == null) {
    return ServiceUtil.returnError("userLogin is null, cannot
                                   create.");
}

Map reviewVals =
    UtilMisc.toMap("userLoginId", userLogin.get("userLoginId"),
                  "planetId", context.get("planetId"),
                  "review", context.get("review"));

GenericValue prGV = delegator.makeValue("PlanetReview",
                                         reviewVals);
prGV.set("reviewId", delegator.getNextSeqId("PlanetReview"));
try{
    prGV.create();
}catch(GenericEntityException e){
    Debug.logError(e, module);
    return ServiceUtil.returnError("An error occurred creating
                                   the PlanetReview record" + e.getMessage());
}

Map result = ServiceUtil.returnSuccess("PlanetReview created");
result.put("reviewId", prGV.get("reviewId"));
return result;
}
```

In the same class add another static method `updatePlanetReview`:

```
public static Map updatePlanetReview(DispatchContext dctx, Map
context){

    GenericDelegator delegator = dctx.getDelegator();
    String reviewId = (String)context.get("reviewId");
    GenericValue planetReview = null;
    Map result = null;

    try{
        planetReview = delegator.findByPrimaryKey("PlanetReview",
                                                UtilMisc.toMap("reviewId", reviewId));
    }catch(GenericEntityException e){

    }
    if (planetReview == null) {
        result = ServiceUtil.returnError("No PlanetReview found
                                       with reviewId of " + reviewId);
        result.put("reviewId", context.get("reviewId"));
        return result;
    }
}
```

```

String originalUserLoginId =
    planetReview.getString("userLoginId");
planetReview.setNonPKFields(context);
planetReview.set("userLoginId", originalUserLoginId);
try{
    planetReview.store();
} catch(GenericEntityException e){
    Debug.logError(e, module);
}
result = ServiceUtil.returnSuccess("PlanetReview updated.");
return result;
}

```

Defining Permission Services

We now declare the permission services themselves, services that the previous two create and update services depend on for permissions checking.

In the file \${component:learning}\servicedef\services.xml, add three new service definitions, of which the first one is an interface implemented by the next two.

```

<service name="permissionInterface" engine="interface">
    <description>Interface to describe base parameters for
        Permission Services</description>
    <attribute name="mainAction" type="String" mode="IN"
        optional="true" />
    <attribute name="resourceDescription" type="String" mode="IN"
        optional="true"/>
    <attribute name="hasPermission" type="Boolean" mode="OUT"
        optional="false"/>
    <attribute name="failMessage" type="String" mode="OUT"
        optional="true"/>
</service>
<service name="learningCheckCreatePlanetReviewPermission"
    engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="checkCreatePlanetReviewPerm">
    <implements service="permissionInterface"/>
    <attribute name="planetId" type="String" mode="IN"/>
</service>

<service name="learningCheckUpdatePlanetReviewPermission"
    engine="java"
    location="org.ofbiz.learning.learning.LearningServices"
    invoke="checkUpdatePlanetReviewPerm">

```

```
<permission-service
    service-name="learningCheckCreatePlanetReviewPermission"/>
<implements service="permissionInterface"/>
<attribute name="reviewId" type="String" mode="IN"/>
<attribute name="planetId" type="String" mode="IN"/>
</service>
```

Notice that the permission service for the update service also relies on a permission service itself. The permission for create checks that the end-user has a postal address on the planet he/she is writing a review for. The permission for update checks that the end-user employs the same user login that created the review being updated. Note how this update permission still needs to check the Planet and PostalAddress constraint, just like that in the create permission. Rather than repeating the code for that check, the update permission relies on the create permission.

Implementing Permission Services

In the class `org.ofbiz.learning.learning.LearningServices`, create a new static method `checkCreatePlanetReviewPerm`:

```
public static Map checkCreatePlanetReviewPerm(DispatchContext dctx,
Map context) {

    String planetId = (String)context.get("planetId");
    GenericValue userLogin =
        (GenericValue)context.get("userLogin");

    List contactMechs = null;
    boolean hasPermission = false;
    try{
        GenericValue party = userLogin.getRelatedOne("Party");
        contactMechs = party.getRelated("PartyContactMech");
        contactMechs = EntityUtil.filterByDate(contactMechs);
        Iterator cmIter = contactMechs.iterator();
        while (cmIter.hasNext()) {
            GenericValue cm = (GenericValue)cmIter.next();
            GenericValue postalAddr =
                cm.getRelatedOne("PostalAddress");
            if (postalAddr != null) {
                if (planetId.equals(postalAddr.get("planetId"))) {
                    hasPermission = true; break;
                }
            }
        }
    }
```

```
        } catch(GenericEntityException e) {
            Debug.logError(e, module);
            return ServiceUtil.returnError(e.getMessage());
        }
        Map result = ServiceUtil.returnSuccess();
        result.put("hasPermission", new Boolean(hasPermission));
        if (!hasPermission){
            result.put("failMessage",
                "You do not have a postal address with planetId of " +
                planetId +
                ". You cannot review a planet you don't live in!");
        }
        return result;
    }
}
```

To fully understand the above code, a fundamental part of the OFBiz data model must be understood. This is based on a widely adopted model as documented by **Len Silverstone** in his book *The Data Model Resource Book*. To fully understand OFBiz, these models must be fully understood.

However, to quickly explain the above code, a `Party` can be a `PartyGroup` (organization, club, company), or a `Person`. Postal addresses and all other contact details (referred to as `ContactMechs`) are related to the `Party` entity, not to the `Person`. A `Party` can have any number of contact details, postal addresses, phone numbers and so we introduce a lookup table called `PartyContactMech`, which holds information on all the `Party`'s contact details. If this record's `contactMechTypeId` is `POSTAL_ADDRESS`, there will be a record in the `PostalAddress` entity. Take a look at the Entity definitions in the `Party` component and try to follow them through. Also take a look at the database structure and records for these entities in the **Entity Data Maintenance** screens from the **Webtools** application.

The above code is simply finding all `PostalAddress` records for the user login and iterating through each one. If the `planetId` field of any of these addresses match the `planetId` that is passed in as a parameter the permission check is successful.

Note how we return in `Map result` the entries `hasPermission` and `failMessage`.

The `ServiceDispatcher` looks at the value in `hasPermission` to determine whether or not our `dispatcher.runSync(...)` code in `createPlanetReview` gets through or not; a boolean value of `true` will allow us through, `false` will not.

If `hasPermission` is `false`, the `ServiceDispatcher` throws an exception with a message that is the value in `failMessage`. This exception is caught by `ServiceEventHandler`, which extracts this message and puts it into request attribute `_ERROR_MESSAGE_`. And this is where we come back again to OFBiz's convention for feedback message placeholders.

Let's implement the other permission service, and then move on to experimentation!

In the same class `LearningServices` create another new static method `checkUpdatePlanetReviewPerm`:

```
public static Map checkUpdatePlanetReviewPerm(DispatchContext
    dctx, Map context) {

    GenericValue userLogin = (GenericValue) context.
        get("userLogin");
    GenericDelegator delegator = dctx.getDelegator();
    Map result = ServiceUtil.returnSuccess();

    GenericValue reviewGV = null;

    try{
        reviewGV = delegator.findByPrimaryKey
            ("PlanetReview", UtilMisc.toMap("reviewId",
                context.get("reviewId")));
    }catch(GenericEntityException e){
        Debug.logError(e, module);
        return ServiceUtil.returnError(e.getMessage());
    }

    if (reviewGV == null) {
        // Leave to service "learningUpdatePlanetReview" to
        // handle this.
        result.put("hasPermission", new Boolean(true));
        return result;
    }

    if (userLogin.get("userLoginId").equals(reviewGV.
        get("userLoginId"))){
        result.put("hasPermission", new Boolean(true));
    }else {
        result.put("hasPermission", new Boolean(false));
        result.put("failMessage", "You cannot update a
            PlanetReview you didn't create!");
    }

    return result;
}
```

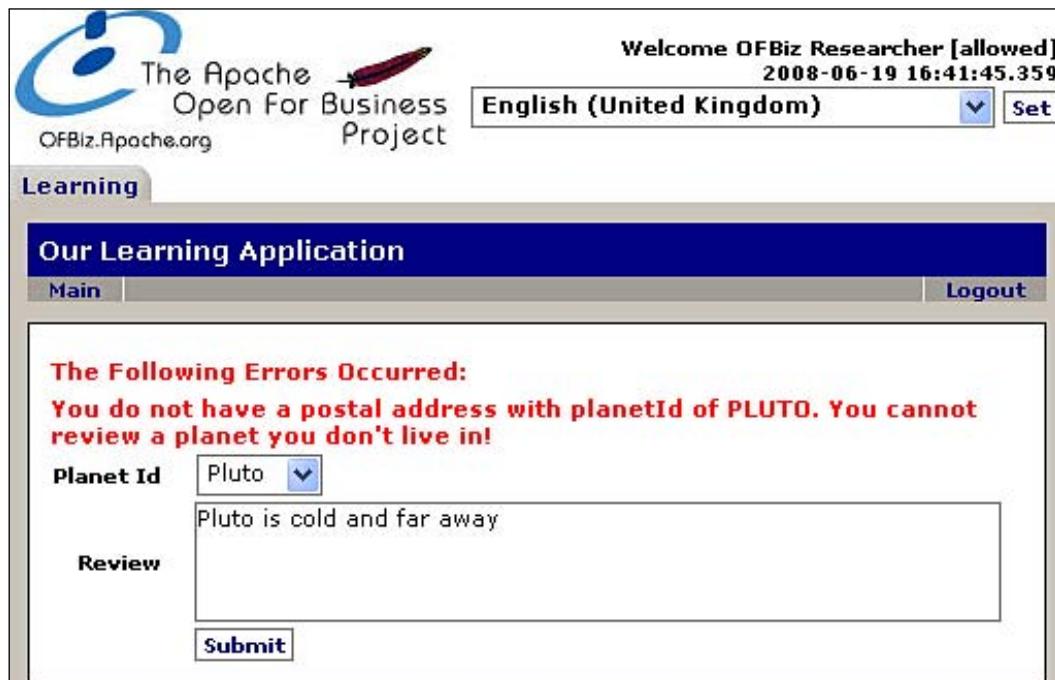
Once all this has been inputted, the application must be stopped, the **Learning** component recompiled and then restarted.

Playing with Complex Permissions

We first need to make sure our Person has a postal address on some planet; there is a chance we might have a postal address with planetId null.

Bring up any postal address our Person has (create one if there are none). Make sure the postal address is on Mars. Also make sure that our Person has no postal addresses on Pluto; we'll be using Pluto as a negative experiment.

Fire to webapp learning an http OFBiz request PlanetReview, and log in as **allowed**. Enter a review for planet **Pluto**, and see our failMessage kick into action. Our Person doesn't have postal addresses on **Pluto**.



The screenshot shows a web application interface for the Apache Open For Business Project. At the top, there's a logo for 'The Apache Open For Business Project' with the URL 'OFBiz.Apache.org'. To the right, a welcome message reads 'Welcome OFBiz Researcher [allowed] 2008-06-19 16:41:45.359' with language settings 'English (United Kingdom)' and a 'Set' button. A navigation bar at the top includes 'Learning' (selected), 'Main', and 'Logout'. Below this is a blue header bar with the text 'Our Learning Application'. The main content area contains a red error message: 'The Following Errors Occurred: You do not have a postal address with planetId of PLUTO. You cannot review a planet you don't live in!'. It shows a dropdown menu set to 'Pluto' and a text input field containing 'Pluto is cold and far away'. On the left, there's a 'Review' section with a 'Submit' button.

Permissions and the Service Engine

Now, write a review for planet **Mars** instead. This time a review for **Mars** can be successfully created. All of **OFBiz Researcher's** addresses have been checked, and one of them is on **Mars**:

The screenshot shows a web application interface for 'Our Learning Application'. At the top, there is a logo for 'The Apache Open For Business Project' with the URL 'OFBiz.Apache.org'. To the right, a welcome message reads 'Welcome OFBiz Researcher [allowed] 2008-06-19 16:49:35.671' with language settings 'English (United Kingdom)' and a 'Set' button. A 'Learning' tab is selected in the navigation bar. The main content area has a blue header 'Our Learning Application' with tabs for 'Main' and 'Logout'. Below this, a section titled 'The Following Occurred:' lists a 'PlanetReview created' event. It details the following information:

Planet Review	created
Review Id	10000
Created By	allowed
Planet Id	Mars
Review	Mars is red and my friends live here.

A 'Submit' button is located at the bottom of the form.

Log out. Fire to webapp learning an http OFBiz request `ListPlanetReviews`, logging in as **denied**:

The screenshot shows a web application titled "Our Learning Application". The main menu has "Main" and "Logout" buttons. Below the menu is a table with columns: Review Id, Created By, Planet Id, Review, and Update. A single row is selected, showing Review Id 10000, Created By allowed, Planet Id MARS, and Review "Mars is red and my friends live here.". An "Update" button is visible in the last column.

Click the **Update** button beside the `PlanetReview` we just created (with user login **allowed**). That brings up the update screen.

Change the field **Planet Id** to **Pluto**, and click **Submit**. We see the first (create) permission service kick-in. The update permission relies on the create permission, so the create permission is activated first. This way we can be sure what if you can't create a review, you can't update one either.

The screenshot shows the "Our Learning Application" update screen. At the top, there's a banner for "The Apache Open For Business Project" and a welcome message "Welcome OFBiz Researcher [denied] 2008-06-19 17:00:30.125". The language is set to "English (United Kingdom)". The main content area has a title "Our Learning Application" and a "Learning" link. It displays an error message: "The Following Errors Occurred: You do not have a postal address with planetId of PLUTO. You cannot review a planet you don't live in!". Below the message are form fields for Review Id (10000), Created By (allowed), Planet Id (Pluto), and Review ("Mars is red and my friends live here."). A "Submit" button is at the bottom.

Change the field **Planet Id** back to **Mars**. Change the review text and click **Submit**. We now see the second (update) permission service kick-in. We got through the create permission, but we are now halted by the update permission. The user login **denied** did not create this **PlanetReview**, **allowed** did. That's why we cannot update this **PlanetReview** while logged in as **denied**.

The screenshot shows a web application interface for 'The Apache Open For Business Project'. At the top, there is a logo and navigation links for 'Learning', 'Main', and 'Logout'. The main content area has a dark blue header bar with the text 'Our Learning Application'. Below this, a red error message is displayed: 'The Following Errors Occurred: You cannot update a PlanetReview you didn't create!'. To the left of the message, there are form fields for 'Review Id' (10000), 'Created By' (allowed), and 'Planet Id' (Mars). The 'Review' field contains the text 'Mars is red and my friends live here.' At the bottom of the form is a 'Submit' button.

Log out. Fire to webapp learning an http OFBiz request `ListPlanetReviews`, logging in as **allowed**. The click on **Update** for the **PlanetReview** we had created before to bring up the **Update** screen.

Change the review text and click **Submit**. Since we created this **PlanetReview** while logged in as **allowed**, we are now able to modify it:

The screenshot shows a web application interface for 'The Apache Open For Business Project'. At the top right, it says 'Welcome OFBiz Researcher [allowed] 2008-06-19 17:12:23.281' with language settings 'English (United Kingdom)' and a 'Set' button. The main navigation bar has 'Learning' selected. Below it, a blue header bar says 'Our Learning Application'. A sub-menu bar has 'Main' selected and 'Logout' on the right. The main content area displays a message: 'The Following Occurred: PlanetReview updated.' It shows details: 'Review Id' is 10000, 'Created By' is 'allowed', and 'Planet Id' is 'Mars'. A dropdown menu also shows 'Mars'. Below this, a text area contains the note: 'Mars is red and my friends live here. Its a bit cold though and the aliens can read my brainwaves.' At the bottom is a 'Submit' button.

Complex Permissions and Simple Permissions cannot be combined

When a `<permission-service>` element co-exists with a `<required-permissions>` element, only the `<permission-service>` element is executed. The `<required-permissions>` element is ignored.

In the file `${component:learning}\servicedef\services.xml`, edit the service definition `learningCreatePlanetReview` to add the following (below the `<permission-service>` element):

```
<required-permissions join-type="OR">
  <check-permission permission="LEARN_VIEW"/>
  <check-role-member role-type="ADMIN"/>
</required-permissions>
```

The user login **denied** should fail the above check because it doesn't have the security permission `LEARN_VIEW` nor the role `ADMIN`.

Restart OFBiz.

Fire to webapp learning an http OFBiz request PlanetReview. Log in as **denied**. Write a new review for **Mars** and submit. The review is created successfully, ignoring the `<required-permissions>` element altogether.

Summary

This concludes our investigation into security and how we can use the Service Engine to determine who has the rights to invoke a service.

In this chapter, we looked at:

- Simple Permissions, Entity Permission, and Role Checks using `<required-permissions>` elements, `<check-permission>` elements, and `<check-role-member>` elements.
- Combining and nesting those permissions and checks using `<required-permissions>` attribute "join-type" and multiple the `<required-permissions>` elements.
- Permission Service (really complex permissions) using `<permission-service>` elements.

In the next chapter, we will look at OFBiz's own language: Minilang and its different uses. Minilang and simple services are the final building block in the foundations of OFBiz, and once understood we will have knowledge of the complete system, and be able to follow any part of the code from the initial request to the final output.

12

Minilang

As the OFBiz framework has evolved over the years, so has the Minilang (Mini-Language). Minilang can help developers to reduce the time it takes to implement simple and repetitive tasks. Code does not need to be compiled and can therefore be implemented faster, breaking the typical Java code-compile-test cycle. Minilang gives the advantage of being able to change the code without a restart of the application. A simple browser refresh is enough to see the changes.

It is more "plain English" than Java code and is simpler to read and therefore easier to understand and maintain by people who may be unfamiliar with the system.

Minilang does however have some drawbacks. It can be difficult to debug in some cases. As we will learn in the next chapter, using Eclipse or other Java IDEs, it is possible to connect to OFBiz using the remote application debugger or the debug tools and step through the Java code line by line, inspecting the value of individual variables. This is not possible with Minilang. Although there are some techniques to help give us visibility to the flow through the code, as the complexity of our services increase, the use of Minilang becomes less and less feasible, and the time taken to debug the code begins to outweigh the benefit of the fast 'hot-deploy' implementation of the code. Minilang is also limited. There are only a certain number of procedures that can be programmed using Minilang.

The main reason for Minilang's existence is to facilitate simple operations, notably CRUD operations and, as we will see in this chapter, to validate and manipulate data. It should not be much used outside of this scope, but within this scope, it excels.

In this chapter we will be looking at:

- Minilang syntax and schema
- Defining and creating a "Simple Service" using Minilang
- Simple events
- Validating and converting fields using the simple-map-processor

- Security in Minilang
- Invoking other services, methods, events, and BeanShell from Minilang
- Using Minilang in screen widgets

What is Minilang?

The syntax of Minilang is simply well formed XML. Developers write XML that obeys a defined schema, this XML is then parsed by the framework and commands are executed accordingly. It is similar in concept to the Gang of Four Interpreter Pattern.

We can therefore consider Minilang's XML elements to be "commands".

Minilang is usually written in a simple method's XML file which is specified at the top of the document like this:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/
    simple-methods.xsd">
```

A local copy of the schema is included with the source code in `framework\minilang\dtd\simple-methods.xsd`.

Although Minilang's primary use is to code services and events, concepts from Minilang are also used to prepare data for screen widgets. We will see towards the end of the chapter.

Much of the simplicity of Minilang arises from the fact that variables are magically there for us to use. They do not have to be explicitly obtained, they are placed in the environment and we can take them as we wish. Should we wish to create a Map, we just use it, the framework will take care of its creation. For example:

```
<set field="tempMap.fieldOne" from-field="parameters.fieldOne"/>
```

will set the value of the `fieldOne` parameter to `tempMap`. If `tempMap` has already been used and is available, this will be added. If not, the Map will be created and the value added to the key `fieldOne`.

Tools to Code XML

Minilang is coded in XML and before it can be successfully parsed by the framework's XML parser, this XML must be well formed. Trying to code Minilang in a plain text editor like Notepad is not a wise move. Precious time can be wasted trying to discover a simple mistake such as a missing closing tag or a misspelled element. For this reason, before attempting to code Minilang services, make sure that you have installed some kind of XML editor and preferably one with an auto-complete feature. The latest versions of Eclipse come packaged with one and XML files are automatically associated to use this editor. Alternatively there are many editors available to download of varying functionality and price. For example XML Buddy (<http://www.xmlbuddy.com>), oXygen XML Editor (<http://www.oxygenxml.com>), or the heavyweight Altova XMLSpy (<http://www.altova.com>).

Defining a Simple Service

Minilang services are referred to as "simple" services. They are defined and invoked in the same way as a Java service. They can be invoked by the control servlet from the `controller.xml` file or from code in the same way as a Java service.

In the following example we will write a simple service that removes Planet Reviews from the database by deleting the records.

First open the file `${component:learning}\widget\LearningForms.xml` and find the PlanetReviews Form Widget. This widget displays a list of all reviews that are in the database.

Inside this Form Widget, immediately under the update field element add:

```
<field name="delete">
<hyperlink target="RemovePlanetReview?reviewId=${reviewId}"
description="Delete"/></field>
```

Our list will now also include another column showing us a hyperlink we can click, although clicking it now will cause an error. We have not added the `request-map` to handle this request in the `controller.xml`. We will add this a little later.

Defining the Simple Service

In the file \${component:learning}\servicedef\services.xml add a new service definition:

```
<service name="learningRemovePlanetReview" engine="simple"
    location="org/ofbiz/learning/learning/LearningServices.xml"
    invoke="removePlanetReview">
    <description>Service to remove a planet review</description>
    <attribute name="reviewId" type="String" mode="IN"
        optional="false"/>
</service>
```

Note the engine type is `simple`.

It is a common practice to group service definitions into their own XML file according to behavior. For instance, we may see that all services to do with Order Returns are in a file called `services_returns.xml`. So long as we add the `<service-resource>` element to the parent component's `ofbiz-component.xml` file and let the system know that this service definition file needs to be loaded, we can structure our service definitions sensibly and avoid huge definition files.

It is not a common practice, however, to group service definitions by type. The type is abstracted from the rest of the system. When the service is invoked, the invoker doesn't care what type of service it is. It could be Java, it could be a simple service, it doesn't matter. All that matters is that the correct parameters are passed into the service and the correct parameters are passed out. For this reason, simple service definitions are found in the same XML files as Java service definitions.

Writing the Simple Method

Simple Method XML files belong in the component's script folder. In the root of \${component:learning} create the nested directory structure `script\org\ofbiz\learning\learning` and in the final directory create a new file called `LearningServices.xml`.

Before we add anything to this file we must make sure that the script directory is on the classpath. Open the file \${component:learning}\ofbiz-component.xml and if it is not already there add

```
<classpath type="dir" location="script"/>
```

immediately underneath the other classpath elements.

The location specified in the service definition can now be resolved.

In our newly created file `LearningServices.xml` add the following code:

```
<simple-methods xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/
        simple-methods.xsd">
    <simple-method method-name="removePlanetReview"
        short-description="Delete a Planet Review">
        <entity-one entity-name="PlanetReview"
            value-name="lookedUpValue"/>
        <remove-value value-name="lookedUpValue"/>
    </simple-method>
</simple-methods>
```

Finally all that is left is to add the `request-map` to the `controller.xml`:

```
<request-map uri="RemovePlanetReview">
    <security auth="true" https="true"/>
    <event type="service" invoke="learningRemovePlanetReview"/>
    <response name="success" type="view" value="ListPlanetReviews"/>
    <response name="error" type="view" value="ListPlanetReviews"/>
</request-map>
```

Since we have added a new service definition OFBiz must be restarted. A compilation is not needed. Restart and fire an http request `ListPlanetReviews` to webapp learning:

Review Id	Created By	Planet Id	Review	Update	Delete
10000	allowed	MARS	Mars is red and my friends live here. Its a bit cold though and the aliens can read my brainwaves.	Update	Delete

Selecting **Delete** will delete this `PlanetReview` record from the database.

Let's take a closer look at the line of code in the simple service that performs the lookup of the record that is to be deleted.

```
<entity-one entity-name="PlanetReview" value-name="lookedUpValue"/>
```

This command will perform a lookup on the `PlanetReview` entity. Since the command is `<entity-one>` the lookup criteria must be the primary key.

This code is equivalent in Java to:

```
GenericValue lookedUpValue = delegator.findByPrimaryKey
    ("PlanetReview", UtilMisc.toMap("reviewId", reviewId));
```

Already we can see that Minilang is less complicated. And this is before we take into account that the Java code above is greatly simplified, ignoring the fact that the delegator had to be taken from the DispatchContext, the reviewId had to be explicitly taken from the context Map and the method call had to be wrapped in a try/catch block.

In Minilang, when there is a look up like this, the context is checked for a parameter with the same name as the primary key for this field, as specified in the entity definition for PlanetReview. If there is one, and we know there is since we have declared a compulsory parameter reviewId in the service definition, then the framework will automatically take it from the context. We do not need to do anything else.

Simple Events

We can call Minilang events, in the same way that we called Java events from the controller.xml. Just as Minilang services are referred to as simple services, the event handler for Minilang events is called "simple".

Tell the control servlet how to handle simple events by adding a new <handler> element to the learning component's controller.xml file, immediately under the other <handler> elements:

```
<handler name="simple" type="request"
        class="org.ofbiz.webapp.event.SimpleEventHandler"/>
```

A common reason for calling simple events would be to perform the preparation and validation on a set of parameters that are passed in from an XHTML form. Don't forget that when an event is called in this way, the HttpServletRequest object is passed in! In the case of the Java events, it is passed in as a parameter. For simple events, it is added to the context, but is nonetheless still available for us to take things from, or add things onto.

In the same location as our LearningServices.xml file (`${component:learning} \script\org\ofbiz\learning\learning`) create a new file called LearningEvents.xml.

To this file add one <simple-method> element inside a <simple-methods> tag:

```
<simple-methods xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/
    simple-methods.xsd">

    <simple-method method-name="simpleEventTest"
        short-description="Testing a simple Event">
```

```

<log level="info" message="Called the Event: simpleEventTest"/>
</simple-method>

</simple-methods>
```

Finally, we need to add a `request-map` to the controller from where this event will be invoked:

```

<request-map uri="SimpleEventTest">
    <security auth=>true</security>
    <event type=>simple</event>
        path=>org/ofbiz/learning/learning/LearningEvents.xml</path>
        invoke=>simpleEventTest/>
    <response name=>success</response> type=>view</type> value=>SimplestScreen/>
    <response name=>error</response> type=>view</type> value=>SimplestScreen/>
</request-map>
```

Notice our simple method doesn't actually do anything other than leave a message in the logs. It is with these messages that we can debug through Minilang.

Validating and Converting Fields

We have now met the Simple Methods Mini-Language, which is responsible for general processing to perform simple and repetitive tasks as services or events. Validation and conversion of parameters are dealt with by another type of Minilang – the Simple Map Processor. The Simple Map Processor takes values from the context Map and moves them into another Map converting them and performing validation checks en-route.

Generally, Simple Map Processors will prepare the parameters passed into a simple event from an HTML form or query string. As such, the input parameters will usually be of type `String`. Other object types can be validated or converted using the Simple Map Processor including: `BigDecimals`, `Doubles`, `FLOATS`, `Longs`, `Integers`, `Dates`, `Times`, `java.sql.Timestamps`, and `Booleans`.

The Simple Map Processors are, like simple methods, coded in XML and they adhere to the same schema (`simple-methods.xsd`). Open this file up again and search for **The Simple Map Processor Section**.

The naming convention for XML files containing Simple Map Processors is to end the name of the file with `MapProcs.xml` (For example, `LearningMapProcs.xml`) and they reside in the same directory as the Simple Services and Events.

One of the best examples of validation and conversion already existing in the code is to be found in the `PaymentMapProcs.xml` file in `${component:accounting}\script\org\ofbiz\accounting\payment`. Open this file and find the simple-map-processor named `createCreditCard`. Here we can see that immediately, the field `expireDate` is created from the two parameters `expMonth` and `expYear` with a "/" placed in between (example, 09/2012):

```
<make-in-string field="expireDate">
    <in-field field="expMonth"/>
    <constant>/</constant>
    <in-field field="expYear"/>
</make-in-string>
```

Towards the end of the `<simple-map-processor>` this `expireDate` field is then copied into the returning Map and validated using `isDateAfterToday`. If the expiration date is not after today, then the card has expired and instead, a `fail-message` is returned.

```
<process field="expireDate">
    <copy/>
    <validate-method method="isDateAfterToday">
        <fail-message message="The expiration date is before today"/>
    </validate-method>
</process>
```

The `<validate-method>` element uses a method called `isDateAfterToday`. This method is in fact a Java static method found in the class `org.ofbiz.base.util.UtilValidate`.

In our examples, we have already been using one of the OFBiz utility classes, `UtilMisc`, namely the `toMap` function, to create for us `Maps` from key-value pairs passed in as parameters. OFBiz provides a huge number of incredibly useful utility methods, ranging from validation, date preparation, and caching tools to String encryption and more. Although some of the most useful utilities will be discussed in Chapter 13, it is worth familiarizing yourself with the methods found in the `UtilValidate` class now.

The framework will automatically allow Minilang access to this class. By adding a bespoke validation method into this class and recompiling, you will be able to call it from the `<validate-method>` in Minilang, from anywhere in your application.

Validating a Simple Event Example

Let's use a Simple Event to create a new `Planet` record and check that the input parameter is not empty. If it is, then return a failure message to the user.

Create a new file named `LearningMapProcs.xml` in `${component:learning}\script\org\ofbiz\learning\learning` and add to it:

```
<simple-map-processors xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/
        simple-methods.xsd">
    <simple-map-processor name="createPlanet">
        <process field="planetId"><copy/><not-empty>
            <fail-message message="Planet Id Cannot Be Empty" />
        </not-empty></process>
        <process field="planetName"><copy/><not-empty>
            <fail-message message="Planet Name Cannot Be Empty" />
        </not-empty></process>
    </simple-map-processor>
</simple-map-processors>
```

In `LearningEvents.xml` add a new simple method:

```
<simple-method method-name="createPlanet"
    short-description="Creating a Planet">
    <call-map-processor xml-resource="org/ofbiz/learning/
        learning/LearningMapProcs.xml"
        processor-name="createPlanet" in-map-name="parameters"
        out-map-name="context"/>
    <check-errors/>
    <make-value value-name="newEntity" entity-name="Planet"/>
    <set-pk-fields map-name="context" value-name="newEntity"/>
    <set-nonpk-fields map-name="context" value-name="newEntity"/>
    <create-value value-name="newEntity"/>
</simple-method>
```

The `<set-pk-fields>` checks the context map returned from the `createPlanet`. function Simple Map Processor checks for any values with a key that have the same name as the primary key of the `GenericValue` object `newEntity` and sets the value to this `newEntity`. In the previous step the `<make-value>` element created a `GenericValue` object in memory from the `Planet` entity.

Likewise the `<set-nonpk-fields>` performs the same action for all other fields. In this case our entity only has two fields. It could be the case that the entity has 20 or more fields. This saves us from explicitly transferring each field from one map to another.

Finally the `<create-value>` element creates a record in the database and persists the populated `newEntity` object.

Minilang

A complete guide to all of Minilang's commands can be found in the *Appendix*. We must now go through the necessary steps of creating screen widgets, Form Widgets and adding request-maps and view-maps to the `controller.xml` file of our component.

To `LearningScreens.xml` add:

```
<screen name="Planets">
  <section>
    <actions><entity-condition entity-name="Planet"
      list-name="planets"/></actions>
    <widgets>
      <decorator-screen name="main-decorator"
        location="${parameters.mainDecoratorLocation}">
        <decorator-section name="body">
          <include-form name="CreatePlanet"
            location="component://learning/widget/learning/
              LearningForms.xml"/>
          <include-form name="Planets"
            location="component://learning/widget/learning/
              LearningForms.xml"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

To `LearningForms.xml` add:

```
<form name="CreatePlanet" type="single" target="CreatePlanet">
  <field name="planetId"><text/></field>
  <field name="planetName"><text/></field>
  <field name="submit" title="Create Planet"><submit/></field>
</form>
<form name="Planets" type="list" target="Planet" list-name="planets">
  <field name="planetId"><display/></field>
  <field name="planetName"><display/></field>
</form>
```

and finally to the learning component's `controller.xml` file add the request-maps:

```
<request-map uri="Planets">
  <security auth=>false< https=>false</>
  <response name=>success< type=>view< value=>Planets</>
  <response name=>error< type=>view< value=>Planets</>
</request-map>
<request-map uri=>CreatePlanet<>>
```

```

<event type=>simple>
    path=>org/ofbiz/learning/learning/LearningEvents.xml>
    invoke=>createPlanet/>
<response name=>success</> type=>view</> value=>Planets</>
<response name=>error</> type=>view</> value=>Planets</>
</request-map>

```

and the single view-map pointing to our screen:

```

<view-map name="Planets" type="screen">
    page="component://learning/widget/learning/
        LearningScreens.xml#Planets"/>

```

Without restarting, fire an OFBiz http request Planets to component learning.

Try entering just a Planet Id or just a Planet Name and see that the validation fails in the Simple Map Processor and the fail-message is returned:

Planet Id	Planet Name
VENUS	
EARTH	Earth
MARS	Mars
PLUTO	Pluto

Finally add a value into both fields and see that the event is executed and the new record is created.

By default the responses `success` or `error` are returned from simple methods. In the above example, the `<check-errors>` element checked the response of the Simple Map Processor and returned `error` if the validation had failed.

Just like the `return String` in a Java event, these responses determine the action of the `request-map`, usually, which `view-map` to use. To change the default response code simply add:

```
<set value="somethingelse" field="_response_code_" />
```

which will correspond to the `request-map` response element:

```
<request-map uri="SomeRequest">
    <event type="simple" path="org/ofbiz/location/SomeEvents.xml"
        invoke="someRequest"/>
    <response name="success" type="view" value="Screen1"/>
    <response name="error" type="view" value="Screen2"/>
    <response name="somethingelse" type="view" value="Screen3"/>
</request-map>
```

Checking Security in Minilang

Permissions can be checked in Minilang by using `<if-has-permissions>`. Since the ecommerce component allows customers to edit and view their own orders, obviously checks need to be made to be sure that customers are not viewing or editing other people's orders. However, we also want customer services staff to be able to edit orders using the same code. Consider the following code:

```
<if>
    <condition>
        <and>
            <not><if-has-permission permission="ORDERMGR"
                action="_VIEW"/></not>
            <if-compare-field field-name="parameters.partyId"
                operator="not-equals"
                to-field-name="userLogin.partyId"/>
        </and>
    </condition>
    <then>
        <string-to-list string="To get order summary
            information you must have the ORDERMGR_VIEW
            permission, or be logged in as the party to get
            the summary information for."
            list-name="error_list"/>
    </then>
</if>
<check-errors/>
```

This checks for the `ORDERMGR_VIEW` permission (our customer services staff). If The user login does not have this permission and the `partyId` of the user login is not equal to the one passed in as a parameter then an error is raised. This error will not cause the Minilang to return until the error is checked for in the `<check-errors/>` command.

Invoking from Minilang

Other services and events can be invoked from both simple services and simple events; again, there is a direct correlation with how other services are invoked from within Java, because ultimately, Minilang is using the Java code.

Calling Services from Minilang

In Java we see that a Map is passed into the service containing all of the service parameters, and a Map is returned:

```
Map resultMap = dispatcher.runSync("ourService", contextMap);
```

Don't forget this is ignoring all necessary error handling, and getting the dispatcher object etc.

In Minilang, we pass a Map in and a Map is passed out, however, the contents of the Map are directly accessible:

```
<call-service service-name="ourService"
              in-map-name="contextMap">
  <result-to-request result-name="exampleId"/>
</call-service>
```

This code calls the service `ourService` with the `contextMap` and takes the `exampleId` OUT parameter from `ourService` and adds it as an attribute on the request. The `userLogin` and `locale` objects are automatically added to the `contextMap`. They do not have to be explicitly set.

Calling Simple Methods

The `<call-simple-method>` command can be thought of as "including" the called simple method code. It is as if the called method code is copied and pasted into the parent simple methods. The environment and context are available to the called simple method:

```
<call-simple-method method-name="inlineMethod"/>
```

Calling Java Methods

The `<call-class-method>` tag allows us to call static Java methods. For example, imagine we wished to directly call the `UtilValidate.isDateAfterToday` method we looked at in the previous example:

```
<call-class-method class-name="org.ofbiz.base.util.UtilValidate"
                   method-name="isDateAfterToday"
                   ret-field-name="booleanIsDateAfterToday">
  <field field-name="expiryDate" type="String"/>
</call-class-method>
```

Calling BeanShell

The `<call-bsh>` tag allows us to put line script into the Minilang for more complicated processing. For example, the following code calculates the promised date of an order and requires manipulation of a `Timestamp`.

```
<set field="daysToShip" from-field="productFacility.daysToShip"/>
<if-empty field-name="daysToShip">
  <set field="daysToShip" value="30" type="Long"/>
</if-empty>
<call-bsh><! [CDATA [
  java.sql.Timestamp orderDate =
    orderHeader.getTimestamp("orderDate");
  java.util.Calendar cal = java.util.Calendar.getInstance();
  cal.setTimeInMillis(orderDate.getTime());
  cal.add(java.util.Calendar.DAY_OF_YEAR, daysToShip.intValue());
  return org.ofbiz.base.util.UtilMisc.toMap("promisedDatetime",
    new java.sql.Timestamp(cal.getTimeInMillis()));
]]></call-bsh>
<set from-field="promisedDatetime"
      field="reserveOisgirMap.promisedDatetime"/>
```

Notice how the `daysToShip` variable is accessible to the inline bsh, and how the `promisedDateTime` is returned in a Map from the inline bsh and is available in the Minilang to be set to the Map `reserveOisgirMap`.

Minilang in Screen Widgets

Minilang concepts are used in screen widgets to prepare data to display on the screen. Minilang commands can be placed within the Screen Widget's `<actions>`. In the Planets example earlier in this chapter we can see the action is:

```
<entity-condition entity-name="Planet" list-name="planets"/>
```

This command performs the lookup on the `Planet` entity with no conditions. All records are therefore returned to the List `planets`.

This `planets` variable is available to all sub-screen and Form Widgets of this Screen Widget. It is therefore available for the Form Widget `Planets`, which iterates through each entry in the list of `GenericValues` and displays the `planetId` and `planetName` fields.

Aside from entity lookups, services can also be invoked from the screen's actions, for example:

```
<service service-name="findParty" auto-field-map="parameters"/>
```

invokes the `findParty` service, automatically populating the input context map from whatever parameters have been passed in from the request. The `OUT` parameters from the service are available to use in the sub-screen and Form Widgets.

There are literally hundreds of existing usages of this concept within OFBiz. It dramatically reduces the need for unnecessary BeanShell action files to prepare data.

Summary

With Minilang, the developer must make a choice. Some developers choose not to increase the already steep learning curve associated with the framework by learning the syntax by heart. Since everything that is possible in Minilang is possible in Java and the fact that Java offers limitless possibilities, while Minilang is limited, many developers choose never to develop in Minilang. However, it is undisputedly easier to code quick and simple services in Minilang than in Java. Especially if you use an auto-completion XML editor. When designing their code, the developer should decide how complex their service is going to be and decide whether to use Java or Minilang.

Whatever the individual preference of the developer, many OFBiz contributors know Minilang intimately and are comfortable coding quite complex services in it. There are many examples existing in the code, and as OFBiz becomes easier and easier to customize, its usage is becoming more and more widespread. To understand the flow through the framework, at the very least a basic knowledge of Minilang must be gained.

Minilang

In this chapter we looked at some fundamental concepts of OFBiz's own language Minilang. In particular we looked at:

- Minilang syntax and schema
- Defining and creating a "Simple Service" using Minilang
- Simple Events
- Validating and converting fields using the simple-map-processor
- Security in Minilang
- Invoking other Services, Methods, Events, and BeanShell from Minilang
- Using Minilang in screen widgets

We should now have built up enough knowledge to find our way through the code from request to output.

Now is a good time to put that to the test!

Open up any of the OFBiz components and click around, enter some details in some forms. Take the request and follow it all the way through the code, looking at any services that are called and understand what they do. This is after all the best way to learn OFBiz!

In the next chapter we will be filling in the remaining few blanks and taking a look at changing the look and feel of OFBiz, FreeMarker and some of the handiest utilities available to us in Tying It Together.

13

Tying Up the Loose Ends

We have seen how the use of screen widgets and Form Widgets provides us with an incredibly quick way of producing XHTML forms, and how we can use them to input and display data.

For back-end office components this is often more than adequate. However, customer facing components may require a more complex and detailed design than Form Widgets can offer. In this chapter we will be taking a look at how we can use FreeMarker to give us a finer degree of control over the layout of the output or even change the output to give us **XSL-FO** and create PDFs.

In this chapter we will study some final techniques that will enable us to set up a customized instance of OFBiz. We'll learn how to:

- Skin OFBiz and change the look and feel
- Use FreeMarker
- Use OFBiz transform tags to aid code reuse
- Use some of the OFBiz utilities
- Call utility methods from FreeMarker
- Output Different Formats

The OFBiz Look and Feel

Over the years the OFBiz look and feel has remained similar. Although the underlying XHTML has changed from a table-based structure to a div-based structure, making it easier for us to control the page layout using stylesheets, the design has remained the same. We are already familiar with the two different looks and feels in OFBiz. Firstly the customer facing ecommerce component:

Tying Up the Loose Ends

The Apache Open For Business Project

Welcome!

Open For Commerce
Part of the Open For Business Family of Open Source Software

Featured Products

Search in Category

Page 1 of 1 **1 - 7 of 7**

Language
English (United States) **Change**

Cart Summary
Shopping Cart is empty

Special Offers

Details Buy 3 Get 2 Free in the Widgets [200] or any sub-category (except the small Widgets [201] category and sub-categories, but always including the Micro Widgets [20111] category), limit to two per order.

Details With special code [9021] get products in the Featured Products category for their special promotion price.

Details Buy 4 items for \$50 from Purple Gizmo [G2-5005], Rainbow Gizmo [G2-1004], Round Gizmo [G2-2644] or Square Gizmo [G2 2002] limit 2 per customer

View All Promotions

Factoids

- Widgets outsell gizmos 2:1
- The use of gizmos has been shown to have no negative effect on personal longevity.

Choose Catalog
Demo Catalog **change**

Search Catalog
Any **All** **Find**
Advanced Search

Browse Categories

- Account Activation
- Configurable PTs
- Gift Cards
- Gift Cards Reload
- Gift Cards Purchase
- Widgets (english)
- Large Widgets
- Small Widgets
- Other Mini Widgets
- Mini Widgets on
- Micro Widgets (english)
- Gizmos (english)
- Large Gizmos
- Small Gizmos

Mouse Hand Poll
Which hand do you use your mouse with?
Right Hand **Submit**

Browse Forums

Browse Content

Gizmos

Financial Account Activation
Balance Account Activation
FA-001 Your Price: From \$1.00 **Choose Variation...**

Gift Card Activation
Give the perfect gift!
CC-001 Your Price: From \$1.00 **Choose Variation...**

Gift Card Reload
Add more money to your card!
GC-002 **Choose Amount...**

Round Gizmo
Round Gizmo with lights - Usually ships in 15 Days!
GZ-2644 List Price: \$48.00 **On Sale!** Your Price: \$38.40
Save: \$9.60 (20%) **Add to Cart**

Configurable PC
Configurable PC
PC001 Your Price: \$50.00 **Configure...**

Tiny Chrome Widget
Tiny Chrome Widget - Usually ships in 2 Days!
WG-5568 List Price: \$60.00 **On Sale!** Your Price: \$48.00
Save: \$12.00 (20%) **Add to Cart**

Giant Widget
Giant Widget with Wheels
Sizes Available: 3-Wheel, 4-Wheel
WG-9943 Compare At: \$22.00 List Price: \$550.00 **On Sale!** Your Price: From \$448.00 Save: \$110.00 (20%) **Choose Variation...**

Page 1 of 1 **1 - 7 of 7**

Secondly the backend office components:

The Apache Open For Business Project

Welcome TIIIC ADMINISTRATOR [admin] 2008-06-29 14:04:51.00

English (United Kingdom) **Set**

Accounting **Catalog** **Content** **Example** **Facility** **Learning** **Manufacturing** **Marketing** **Order** **Party** **WebTools** **WorkEffort**

Logout

Party Manager Application

Main **Find** **Create** **Link Party** **Communications** **Visits** **Classifications** **Security** **Address Match Map**

Find Party

Contact Information : None Postal Telecom Other

Party ID :

User Login :

Last Name :

First Name :

Party Group Name :

Role Type : Any Role Type **Lookup Party** **Show all records**

Parties Found
No parties found. **Create New**

W3C CSS **W3C XHTML 1.0**

Copyright (c) 2001-2008 The Apache Software Foundation - www.apache.org
Powered by Apache TIIIC Release 4.0

Out of the box, the look and feel for both are similar. But, they purposefully use different stylesheets and decorators so the styles and structure of each can be changed, independent of one another.

The primary **Cascading Style Sheet (CSS)** for the ecommerce component can be found in the directory `${ofbizInstallFolder}\framework\images\webapp\images` and is called `ecommain.css`.

This image component is in the framework directory since it supplies images, stylesheets and JavaScripts throughout the entire application. If we upload a product image through the product screens, it is stored here. For large shops with a wide product range, this directory could end up being very large and, as we will learn later in the chapter, OFBiz has made it easy for us to move this directory to a different location, even onto a different dedicated image server if we so wish.

Start OFBiz and open the ecommerce component by navigating your browser to `http://localhost:8080/ecommerce`. View the source of the home page.

We can see that design of the XHTML source is `<div>` based. This structure is ultimately controlled by the screen widget `main-decorator` located in the file `${component:ecommerce}\widget\CommonScreens.xml`.

The ecommerce component has been specifically designed to be as configurable as possible and made as easy for us to change to our needs as possible. As such, every effort has been made to stop us from needing to delve into the code and change this structure. The location of the logo and the location of the CSS are stored within the `ProductStore` entity in the database.

Changing the Customer Facing Site

OFBiz is capable of serving different websites with different product stores, each having different categories and products. By default, the ecommerce component is "bound" to the `Website` record with the `websiteId` of value `WebStore` by a context parameter in the ecommerce webapp's `web.xml` file. Because of this context parameter, the ecommerce application knows which `ProductStore` data to load.

Take a look at how this `Website` can be configured by opening the **Content Manager Application** at `https://localhost:8443/content/control/main` and clicking on the **WebStore** link.

From here, the `http` and `https` hosts and ports that the customer facing website listens on can be changed, giving us the ability to access more than one site. In the next chapter we will learn how to configure Apache to handle this more elegantly, so the customer can access each different site using different url's, rather than using the same url and changing the port.

Tying Up the Loose Ends

Notice that the `productStoreId` field is set to the **OFBiz E-Commerce Store**. Select the **Catalog** tab from the tab menu at the top of the page, select **Stores** and then **OFBiz E-Commerce Store [9000]**.

The `ProductStore` entity fields allow us to control and quickly change many aspects of our store. Most are self explanatory. For the purpose of design configuration we are going to change the **Store Name**, **Company Name**, **Title**, and **Sub-Title** fields to the following:

Store Name	Our Learning Store
Company Name	Learning Inc.
Title	Customer Facing ProductStore
Sub-Title	Part of the Learning OFBiz Example

and half-way down the form change the **Style Sheet** and **Header Logo** to:

Style Sheet	/images/learning.css
Header Logo	/images/learning_logo.jpg

Select **Update**. Our changes will not yet be visible. Since this record in the `ProductStore` entity is so commonly accessed (don't forget, every person who visits the site will perform a lookup on this record), it is cached. Clear the cache by selecting **WebTools** from the top tab menu, then select **Cache Maintenance** and finally **Clear All Caches**.

Return to the ecommerce site and press *Ctrl+F5* to ensure that our browser pulls a fresh copy of the site from the server and not from its local cache. Unsurprisingly, all we see is text with no formatting. We have not yet created the stylesheet.

Create a simple logo with a height of 69 px using a graphics package and save as `learning_logo.jpg` in `${component : images} \webapp\images`. for example:



Finally, in the same directory, create a new file called `learning.css`, and copy into it the entire contents of the file `ecommaint.css`.

Now we can have some fun! Keeping the ecommerce application open in the browser in the background, perform a **Find and Replace All** (*Ctrl+F* in Eclipse) in the `learning.css` file and change all instances of `#000099` to `#009933`. This should change the OFBiz color scheme from **blue** to **green**. Return to the browser and press **F5** to refresh and see our modified store.

The screenshot shows the Apache OFBiz Customer Facing ProductStore. The top navigation bar includes links for Login, Contact Us, Main, Quick Add, Order History, Shopping Lists, Requests, Quotes, and Profile. A banner at the top reads "Customer Facing ProductStore Part of the Learning OFBiz Example Welcome!" and "Shopping Cart is empty [View Cart] [Quick Checkout]". On the left, there's a sidebar with sections for Choose Catalog (Demo Catalog), Search Catalog (Advanced Search), Browse Categories (listing various widget types like Account Activation, Configurables PCs, etc.), Mouse Hand Poll (asking which hand you use your mouse with), and Browse Forums. The main content area displays a grid of featured products:

- Financial Account Activation**: Balance Account Activation FA-001 Your Price: From \$1.00
- Gift Card Activation**: Give the perfect gift! GU-001 Your Price: From \$1.00
- Gift Card Reload**: Add more money to your card! GC-002
- Round Gizmo**: Round Gizmo with lights - Usually ships in 15 Days! GZ-2644 List Price: \$48.00 On Sale! Your Price: \$38.40 Save: \$9.60 (20%)
- Configurable PC**: Configurable PC PC001 Your Price: \$50.00
- Tiny Chrome Widget**: Tiny Chrome Widget - Usually ships in 2 Days! WG-5569 List Price: \$60.00 On Sale! Your Price: \$48.00 Save: \$12.00 (20%)
- Giant Widget**: Giant Widget with Wheels Gizes Available: 3 Wheel, 4 Wheel WG-9943 Compare At: \$922.00 List Price: \$550.00 On Sale! Your Price: From \$440.00 Save: \$482.00 (20%)

On the right side, there are sections for Language (set to English (United States)), Cart Summary (empty), Special Offers (details about discounts for purchases over \$1000 and specific items), and Factsoids (information about widget resale value and popularity). A "Last Categories" section is also present.

Obviously, this is the most basic change in design that is possible. How far you want to change the design is up to you. To gain a feel for how customizable the customer facing application is, take a look at the *Who Is Using Apache OFBiz* section of the OFBiz site (<http://www.ofbiz.org>).

Changing the Back Office Screens

The back office screens can also be customized to give OFBiz your corporate identity. Obviously, however, having a web designer completely change the design of the back office screens could be a waste of money. Customers should never see these screens; therefore the design of these screens is not as critical as those in the ecommerce component.

We have already seen how the main decorator `GlobalDecorator` is shared throughout the entire back office and is found in `${ofbizInstallFolder}\framework\common\widget\CommonScreens`.

By reading through this decorator, it becomes very easy to see how to customize these screens.

Each section of the screen is broken up into four main sections. The header, the appbar, the center area, and the footer.

The main style for these screens is in the `maincss.css` file in `framework\images\webapp\images`.

The Header

The Header displays the logo, login and locale information to the user.

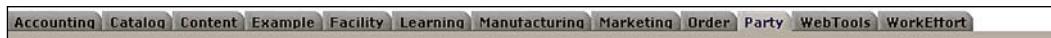


Location: `${component:common}\webcommon\includes\header.ftl`

The `header.ftl` file can be found in the common component. If you were to change the logo location within this file the change will be apparent across all of the back office screens.

The Applications Bar (appbar)

The appbar's job is to display tabs at the top of the back-office screens allowing us to quickly enter other components.



Location: `${component:common}\webcommon\includes\appbar.ftl`

By default, the tabs will appear. By adding the attribute `app-bar-display="false"` to the `webapp` element, they will be suppressed. For example, to stop the ecommerce component from appearing:

```
<webapp name="ecommerce"
        title="eCommerce"
        server="default-server"
        location="webapp/ecommerce"
        mount-point="/ecommerce"
        app-bar-display="false"/>
```

The base permission from the webapp is also compared with the logged in user's permissions. If the logged in user's permissions meets or exceeds the base permission, the tab will be displayed:

```
<webapp name="party"
    title="Party"
    server="default-server"
    location="webapp/partymgr"
    base-permission="OFBTOOLS, PARTYMGR"
    mount-point="/partymgr"/>
```

The Central Area

The screenshot shows the 'Party Manager Application' interface. At the top, there is a navigation bar with links: Main, Find, Create, Link Party, Communications, Visits, Classifications, Security, Address Match Map, and Logout. Below the navigation bar is a search form titled 'Find Party'. The form has fields for 'Contact Information' (with 'None' selected), 'Party ID' (text input), 'User Login' (text input), 'Last Name' (text input), 'First Name' (text input), 'Party Group Name' (text input), and 'Role Type' (dropdown menu set to 'Any Role Type'). Below the form are two buttons: 'Lookup Party' and 'Show all records'. At the bottom of the central area, there is a message box titled 'Parties Found' containing the text 'No parties found.' and a link 'Create New'.

The central area of the screen is component-specific and changes to display the information and data that you wish to see. This area can be again broken down into smaller areas, including the main body area, where our forms and data are displayed.

The Application Header



Location: \${component:xxxxx}\webapp\xxxx\includes\appheader.ftl

This contains the main title for the component and top level links that are at the moment important destinations for this component. They are visible from most (if not all) pages within the component.

The Footer

Finally, The Footer displays copyright information, or any other information we wish to display on every page.



Location: \${component:common}\webcommon\includes\footer.ftl

Once again, a common shared file where any changes are propagated throughout the entire back office.

Using FreeMarker

FreeMarker is another Open Source project. It is a template engine which has some programming capabilities. For those who are familiar with Velocity (another templating engine), the migration to FreeMarker is easy, as they share many of the same concepts. OFBiz makes great use of FreeMarker's flexibility to display data that has been returned from event and services or lookups performed in the screen widgets actions in different formats, usually XHTML. The separation of the presentation into these FreeMarker templates allows web designers to concentrate solely on the design and layout and keeps their need to know any other programming languages to a minimum. Freemaker Template Language is commonly referred to as FTL, and FreeMarker templates have the file extension .ftl.

Throughout the course of this book, we have used some FreeMarker templates already. The messages.ftl file that has been used in so many of the examples is a very good example of such a file. Let's remind ourselves how it is included in the screen widgets:

```
<platform-specific>
  <html>
    <html-template location="component://common/webcommon/
      includes/messages.ftl"/>
  </html>
</platform-specific>
```

Using FreeMarker, we are able to iterate through lists passed to us from actions that have prepared data and perform some simple programming tasks to help us manipulate and display data.

To demonstrate this let's take a look at the screen that prepared and displayed for us the list of planet reviews in our LearningScreens.xml file:

```
<screen name="ListPlanetReviews">
  <section>
    <actions><entity-condition entity-name="PlanetReview"
      list-name="planetReviews"/></actions>
    <widgets>
      <decorator-screen name="main-decorator"
        location="${parameters.mainDecoratorLocation}">
        <decorator-section name="body">
          <include-form name="PlanetReviews"
            location="component://learning/widget/
              learning/LearningForms.xml"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

Here we allowed the form widget to perform the iterative work for us, cycling through each element in the planetReviews List that was obtained from the **<entity-condition>** lookup.

Fire an http request ListPlanetReviews to the webapp learning. If you do not have any reviews listed here first send a request to PlanetReview and, logged in as **allowed**, create some test reviews before any changes are made. This way there will definitely be some reviews for our FreeMarker template to display.

In the ListPlanetReviews screen, replace the line:

```
<include-form name="PlanetReviews"
  location="component://learning/widget/learning/
    LearningForms.xml"/>
```

with our FTL file. Include:

```
<platform-specific>
  <html>
    <html-template location="component://learning/webapp/
      learning/planets/reviews.ftl"/>
  </html>
</platform-specific>
```

And finally create a new directory \${component:learning}\webapp\learning\planets and in it add a new file reviews.ftl with the contents:

```
<#assign index = 0 />
<table cellspacing="0" class="basic-table">
  <tr class="header-row">
```

```
<td>Review Id</td>
<td>Created By</td>
<td>Planet Id</td>
<td>Review</td>
<td>Update</td>
<td>Delete</td>
</tr>
<#list planetReviews as review>
<form name="reviewForm_o_${index}" action="<@ofbizUrl>PlanetReview</ofbizUrl>" method="post">
<input type="hidden" name="reviewId" value="${review.reviewId}" />
<tr>
<td>${review.reviewId}</td>
<td>${review.userLoginId}</td>
<td>${review.planetId}</td>
<td>${review.review}</td>
<td><input type="submit" name="submit" value="Update" /></td>
<td><a href="<@ofbizUrl>RemovePlanetReview?reviewId=${review.reviewId}</ofbizUrl>">Remove</a></td>
</tr>
</form>
<#assign index = index + 1 />
</#list>
</table>
```

We have now replicated the same functionality given to us in the form widget. The FTL, though more complicated to code, produces slightly simpler XHTML since the JavaScript used to submit the forms in the form widget has been ignored and replaced with a standard **Submit** button.

Notice the forms action and also the RemovePlanetReview links are surrounded by `<@ofbizUrl></@ofbizUrl>` tags. These transform tags resolve the correct URLs for us.

OFBiz Transform Tags

There are a number of incredibly handy and widely used bespoke FreeMarker tags which as the template is rendered, transforms the contents of the tags for us.

`<@ofbizUrl>`

This tag will resolve the action in the above forms from

```
<@ofbizUrl>PlanetReview</ofbizUrl>
```

to:

```
/learning/control/PlanetReview
```

By allowing the framework to resolve these tags for us, we are able to completely ignore within the templates whether the links should point to `https://` or `http://`. This can now be controlled and changed in just one place, in the `controller.xml` request-map's `<security>` elements.

Port and host name settings are checked first from the `Website` entity and then from the `url.properties` (`${ofbizInstallFolder}\framework\webapp\config`) file. These settings are checked on the resolution of every link to ensure the link is pointing to the correct place.

The transformation adds the correct webapp name, the control path and the desired request. This allows us to reuse screens widgets or FreeMarker templates from their existing locations without having to import a copy into our component and then having to change all the links. All we have to do is reference them from within our screen widgets and ensure that our `controller.xml` file has corresponding request-maps. Take a look through some of the screen widget files in the ecommerce component and see how many `.ftl` files from the order component are referenced.

<@ofbizContentUrl>

The principle behind this transform tag is similar to `<@ofbizUrl>`. It will resolve all links to point to the correct image location. By default, the images folder containing all JavaScripts, stylesheets, and all images belongs in `${ofbizInstallFolder}\framework\images`. However, with the product images being included in this directory, along with all the design content, this directory could grow very large. We may be running multiple instances of OFBiz on different servers and want to ensure that the images are all uploaded to and served from the same location. In this case we would need to move this images directory to its own dedicated image server. So long as every time we reference an image from the FreeMarker templates using the `<@ofbizContentUrl>` tags we can tell the framework in just one place that the location of the images folder has changed. This is the `content.url.prefix` standard property found in the `url.properties` file.

To include an image in the FreeMarker template, the `src` would become:

```

```

Using these transform tags is a good practice and is a good habit to keep from the start. It can save hours of trawling through code changing every reference once you decide to change the system architecture.

<@ofbizCurrency>

This transform tag prepares and displays currency data for us, stopping us from having to hard code currency symbols. This tag formats the number passed in and rounds it to the correct number of decimal places. To show the amount **16.45** in US Dollars we would use the code:

```
<@ofbizCurrency amount=16.45 isoCode="USD" />
```

If you wish to completely understand what these transform tags do and how they work, take a look at the `OfbizUrlTransform`, `OfbizContentTransform`, and `OfbizCurrencyTransform` classes in the package `org.ofbiz.webapp.ftl`.

Calling Java from FreeMarker

The `FreeMarkerViewHandler` prepares an environment that enables us to use the OFBiz objects that were made available for us in the services and events:

- `delegator`
- `dispatcher`
- `security`
- `userLogin`
- `session`
- `request`

To make life even easier, we can access any request parameters or attributes using:

- `requestParameters`
- `requestAttributes`
- `sessionAttributes`

No further actions must be performed to use these objects from the FreeMarker templates. For example, to display a parameter `parameterOne`, that has been passed into the request we would write:

```
 ${requestParameters.parameterOne}
```

The framework also prepares the environment such that it allows us to call static Java methods directly from the FTL.

For example, to prepare a Map and assign it to a FreeMarker variable we could put:

```
<#assign exampleMap = ${Static["org.ofbiz.base.util.UtilMisc"].toMap  
("keyOne", "valueOne", "keyTwo", "valueTwo") } />
```

If we then wished to see the contents of this `exampleMap`, then we just put `${exampleMap}` in the template. By doing that, the entire content is very conveniently written out to the screen.

This can be a very handy way to debug FreeMarker and quickly examine the contents of the variables.

This becomes particularly useful when examining the contents of a looked up entity record. To see this in action in `reviews.ftl` immediately under the line:

```
<#list planetReviews as review>  
add:  
  
<#assign userLogin = review.getRelatedOne("UserLogin") />  
<tr><td colspan="6">${userLogin}</td></tr>
```

and refresh the page. We should now see each `UserLogin` entity record written out in full above the record.

Had the `UserLogin` entity not been a foreign key relation of `PlanetReview`, we would have had to perform a `delegator.findByPrimaryKey` lookup to find the record. For example:

```
<#assign userLogin = delegator.findByPrimaryKey("UserLogin",  
        Static["org.ofbiz.base.util.UtilMisc"].toMap("userLoginId",  
        review.userLoginId)) />
```

OFBiz Utilities

OFBiz contains a large number of very useful Utility classes that can be called from anywhere throughout the application, including FreeMarker templates.

Some of the most useful ones are contained in the package `org.ofbiz.base.util`. Some of their uses include String manipulation, validation, encryption, debugging, and date manipulation.

It is worth spending some time becoming familiar with the methods found in these classes. Very often the simple challenges we face have already been overcome by somebody else.

The following section is a guide to a few of the most useful and most commonly used methods from some of these Java classes.

UtilMisc

These miscellaneous utility methods are generally to aid List or Map retrieval, iteration or creation.

```
Map returnMap = UtilMisc.toMap("keyOne", "valueOne");
```

A map of up to six key value pairs can be created in this way.

```
List returnList = UtilMisc.toList("valueOne", "valueTwo");
```

Again a List of up to six values can be created in this way.

UtilValidate

Validation methods can save time and code by performing validation tests on Strings, Maps, Lists, or other Objects

To check if a variable is empty.

```
boolean isEmpty = UtilValidate.isEmpty(returnMap);
```

or

```
boolean isNotEmpty = UtilValidate.isNotEmpty(returnMap);
```

In the case of Lists and Maps, a null value is checked for first, followed by a check for an empty instantiated object with no elements.

UtilDateTime

This class contains handy `java.sql.Timestamp` and `java.util.Date` manipulation methods.

To return a `java.sql.Timestamp` object to the nearest millisecond:

```
Timestamp now = UtilDateTime.nowTimestamp();
```

Debug

These widely used logging utility methods allow us to use log4j to quickly write to the logs. The level of logging to be displayed is determined in `${ofbizInstallFolder}\framework\base\config\debug.properties`:

```
Debug.logError(e.getMessage(), module);
```

Outputting Different Formats

So far we have only seen one type of View Handler: the `ScreenWidgetViewHandler`, which renders screen widgets as XML, usually XHTML.

Sometimes we may need to change the output to something else. We may just wish to produce a formatted report of all sales this month that can be e-mailed or saved, or we may wish to have the system produce shipping label files that can be automatically sent to a printer and a copy saved to the hard disk.

Outputting a PDF

Using similar screen widget concepts to those we have already met, it is possible to produce PDFs using the `screenfop` View Handler, `ScreenFopViewHandler`.

Formatting Objects Processor (FOP) is another Apache top level project and is a print formatter that can produce PDFs from XSL-FO (Extensible Stylesheet Language—Formatting Objects). In essence, we use screen widgets to produce XSL-FO which is actually just a different type of XML, which is then formatted into a PDF by FOP.

This is not as complicated as it sounds and much of this work is performed in the background by the framework. The FOP libraries, like the FreeMarker libraries, are already included in the OFBiz project and the `ScreenFopViewHandler` is already written and in use throughout the code. All we have to do is create request-maps, view-maps, and a new ftl file which will render the XSL-FO.

Open the learning component's `controller.xml` file, and underneath the `screen` `<handler>` element add:

```
<handler name="screenfop" type="view"  
        class="org.ofbiz.widget.screen.ScreenFopViewHandler"/>
```

We have now told the control servlet how to handle this new type of view.

Add the request-map and the view-map to the same file in the appropriate places (underneath the `ListPlanetReviews` mappings).

```
<request-map uri="ListPlanetReviewsPDF">  
    <security auth="false" https="false"/>  
    <response name="success" type="view" value="ListPlanetReviewsPDF"/>  
</request-map>  
  
<view-map name="ListPlanetReviewsPDF" type="screenfop"  
        page="component://learning/widget/learning/  
              LearningScreens.xml#ListPlanetReviewsPDF"  
        content-type="application/pdf" encoding="none"/>
```

The view-map's content-type and encoding attributes are used to inform the browser what type of file has been received. If the browser is configured properly and application/pdf types are associated with Adobe Acrobat Reader or a similar installed PDF reader, streams of this content-type will be automatically opened using this application.

To `LearningScreens.xml` add the new `ListPlanetReviewsPDF` screen widget:

```
<screen name="ListPlanetReviewsPDF">
    <section>
        <actions><entity-condition entity-name="PlanetReview"
            list-name="planetReviews"/></actions>
        <widgets>
            <platform-specific>
                <html><html-template location="component://learning/
                    webapp/learning/planets/reviews.fo.ftl"/></html>
            </platform-specific>
        </widgets>
    </section>
</screen>
```

Even though this actually generates XSL-FO, the HTML oriented render will output the XSL-FO correctly, so our new FreeMarker template is called as if it were an HTML template.

This screen widget does not use a decorator. All of the information needed to produce the PDF will be contained in the file `reviews.fo.ftl`. It is possible to wrap the screen widgets in a decorator if many of them are using the same chunk of XSL-FO over and over again.

Next, in the directory `learning\webapp\learning\reviews` add the FTL `reviews.fo.ftl` and to it add:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
        <fo:simple-page-master master-name="review-page">
            <fo:region-body margin="1in"/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="review-page">
        <fo:flow flow-name="xsl-region-body">
            <fo:block>
                <fo:table text-align="center" table-layout="fixed">
                    <fo:table-column column-width="2.00cm"/>
```

```
<fo:table-column column-width="4.00cm"/>
<fo:table-body>
    <fo:table-row>
        <fo:table-cell>
            <fo:block line-height="6pt"
                      space-before.optimum="1.5pt"
                      space-after.optimum="1.5pt"
                      keep-together="always">
                <fo:inline font-size="6pt">Review
                    Id</fo:inline>
            </fo:block>
        </fo:table-cell>
        <fo:table-cell>
            <fo:block line-height="6pt"
                      space-before.optimum="1.5pt"
                      space-after.optimum="1.5pt"
                      keep-together="always">
                <fo:inline font-size="6pt">
                    Review</fo:inline>
            </fo:block>
        </fo:table-cell>
    </fo:table-row>
    <#list planetReviews as review>
        <fo:table-row>
            <fo:table-cell>
                <fo:block line-height="6pt"
                          space-before.optimum="1.5pt"
                          space-after.optimum="1.5pt"
                          keep-together="always">
                    <fo:inline font-size="6pt">
                        ${review.reviewId}</fo:inline>
                </fo:block>
            </fo:table-cell>
            <fo:table-cell>
                <fo:block line-height="6pt"
                          space-before.optimum="1.5pt"
                          space-after.optimum="1.5pt"
                          keep-together="always">
                    <fo:inline font-size="6pt">
                        ${review.review}</fo:inline>
                </fo:block>
            </fo:table-cell>
        </fo:table-row>
    </#list>
</fo:table-body>
</fo:table>
</fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>
```

Finally fire an http request `ListPlanetReviewsPDF` to webapp learning to see the PDF produced.

XSL-FO can be tricky to learn and can also be an unforgiving language. Whereas unclosed table row tags (`</tr>`) may not necessarily effect the way the browser displays the page, or if there is an error in the HTML, more often than not something will at least be displayed giving some clue as to what needs fixing. With FOP an unclosed tag will often result in an error producing nothing. It can be difficult to debug and time consuming.

There are other types of View Handlers already written and available to use including Jasper Reports, JSP, and Velocity (see the package `org.ofbiz.webapp.view`). With some study of the existing View Handlers it becomes quite simple to write your own should you need to output other formats.

Summary

In this chapter, we have taken a look at some of the final missing pieces of our knowledge about OFBiz and gone a few steps further by changing the output from XHTML to something else entirely.

We now have the knowledge to change the look and feel of both the customer facing pages and the back office pages

We have seen how powerful and versatile FreeMarker can be and how we can use the OFBiz transform tags to help us reuse FreeMarker templates from one component in another component without duplication.

In this chapter we have:

- Skinned OFBiz and changed the look and feel
- Used Freemarker with OFBiz transform tags
- Used some of the OFBiz utilities
- Called utility methods from Freemarker
- Outputted a simple PDF report

Now that we are drawing to the end of our learning journey, it may be time to consider looking at the steps that are necessary or recommended by other OFBiz users to prepare our instance for production. The Apache OFBiz Technical Production Setup guide is a must read. In this document you will find instructions on how to change the ports to the standard port 80 and 443, change the debug and cache settings (changing these will have a massive impact on performance), set up security and install an SSL certificate. This document can be found at <http://docs.ofbiz.org/display/OFBTECH/Apache+OFBiz+Technical+Production+Setup+Guide>.

In the next chapter we will be taking a look at some advanced techniques with OFBiz, including how to manage a project and keep up to date with new bug fixes and features from the repository. We'll also see some architectural design ideas as well as seeing how to run OFBiz behind an Apache HTTP server.

14

Tips and Techniques

Now that we have the necessary skills and understand OFBiz enough to comfortably develop our own custom applications or tailor existing ones to our business needs, this chapter focuses on some more advanced topics including some critical time saving debugging and development techniques and some hints on how to manage a project with a team of developers using SVN to help maintain an up-to-date code base.

One common architectural design is to run OFBiz behind the widely used and stable Apache HTTP Server. Although OFBiz comes bundled with Tomcat and can run out of the box, serving web pages and handling **http requests** and **placing a web server** in front of OFBiz gives a great deal of flexibility. This allows us to set up different instances of OFBiz on one server, or the same instance of OFBiz on multiple servers and load balance between them. In this chapter we will be exploring setting up the Apache HTTP Server and connecting between the web server and OFBiz.

In this chapter we will be studying:

- Debugging techniques
- Maintaining a project with the latest bug fixes and features
- Working with the latest version
- Running OFBiz with Apache HTTP Server using `mod_proxy_ajp` connector
- Development tips

Debugging Techniques

Since the OFBiz framework is comprised of a number of different concepts and programming languages (Java, Minilang, FreeMarker, BeanShell) there is no single technique to debug the code. To gain visibility at run-time, a number of different techniques must be applied, ranging from the full debug capabilities of Eclipse or another IDE, to outputting messages and variable values to the logs or even the screen.

Logging

In the early chapters we used the logs to check what was happening when we started OFBiz and installed the seed data.

OFBiz writes to a number of different logs which are stored in the directory `${ofbizInstallFolder}\runtime\logs`.

The `access_log` files are similar to the access log files of the Apache HTTP Server and contain every request made to the server. These are handy to archive and can be used by programs such as *AWStats* (<http://awstats.sourceforge.net>) to produce graphical information on the site's traffic activity.

console.log

By default, OFBiz writes the contents of the console to this file. However, we may want to watch for an error in real time, and so choose to write the contents of the console to the console and watch the logs scroll up the window. This is achieved by editing the `startofbiz.bat` file and removing the `> runtime\logs\console.log`.

ofbiz.log

This is the main log in OFBiz and contains all logging messages generated by the framework. Once it is full, it is renamed `ofbiz.log.1` and `ofbiz.log` starts from empty. Each log is full when it reaches 1000KB (roughly 1MB) and a total number of 10 backup files are stored. Writing to a small log file is more efficient than writing to a large one so it is better to keep the log file down to a reasonable size for performance issues. Depending on the site traffic 10MB of logs may be sufficient, although this is configurable.

Configuring the Logs

How much information these logs display is configurable and the level of information displayed is controlled in the file `debug.properties`, in the directory `framework\base\config`.

By default, the only level that is switched off is `print.verbose`. Notice that the value is `ftrue`. It doesn't have to be set to `false` to be false. It can be any value but `true`. Try switching this to `true` and restarting. Verbose logging displays an enormous amount of information, stretching all the way back to entity engine activities, showing queries and prepared statements, database pooling, and transactional information. The more information that is written to the logs, the longer it takes to create these logs, and having the debugging levels set to display too much information will have serious performance consequences. It is recommended that for production environments everything below `warning` is switched off. This means that only fatal errors, errors, and warnings are displayed.

To change the size of the `ofbiz.log` files and the number of historical backup files that are stored change the values of the following properties:

```
log4j.appenders.file.MaxFileSize=1000KB  
log4j.appenders.file.MaxBackupIndex=10
```

A considerable amount of work has been undertaken on the logging since OFBiz 4.0 was released. In the next release of OFBiz we can look forward to a greater visibility, with the ability to write log levels into their own files rather than all lumped into one large file.

Viewing the Logs through Webtools

The `log4j.appenders.css` writes logs in an XHTML format to a file `ofbiz.html`. This file is accessible through the **Webtools** console and is displayed to us neatly formatted. This useful feature eliminates developers needing server root access just to gain access to the logs.

To find the logs, open up the **Webtools** console (<https://localhost:8443/webtools/control/main>) and select **View Log**.

The logging level can also be changed just for one session through the **Webtools** console by selecting **Adjust Debugging Levels**.

Writing Information to the Logs

In Java code we have already briefly met the handy OFBiz utility class `Debug`. We can make use of this to write the information we wish to log at any particular level.

For example:

```
Debug.logVerbose("This is a Verbose Logging Message", module);
```

The `module` parameter passed in is a static class variable with a value of the name of the class. This gives the logger the information it needs to tell us where the log message came from. For example:

```
public static String module = LearningEvents.class.getName();
```

The following methods are the most commonly used:

```
Debug.logVerbose("Message", module);
Debug.logInfo("Message", module);
Debug.logImportant("Message", module);
Debug.logWarning("Message", module);
Debug.logError("Message", module);
Debug.logFatal("Message", module);
```

We may wish to determine a course of action and perform a lookup or an extra function depending on whether a particular logging level is on. For example, we may wish to display (in the logs) a person's name instead of just their `partyId`, but only when verbose logging is on. Let's say that this person's name is not to hand and requires another lookup. We can perform a very quick check using the `Debug.verboseOn()` method to determine if this logging level is on. So:

```
if(Debug.verboseOn()) {
    try{
        GenericValue person = delegator.findByPrimaryKey
            ("Person", UtilMisc.toMap("partyId", partyId));
        Debug.logVerbose("This is person : " +
                        person.getString("firstName") + " " +
                        person.getString("lastName"), module);
    }catch(GenericEntityException e){
        // Handle this error
    }
}
```

Logging Minilang

Logging is the only way to currently debug through Minilang and again can be restricted by level:

```
<log level="info" message="This is the Debug Message - ${someField}" />
```

The levels are **verbose**, **info**, **important**, **warning**, and **error**.

Since often these log messages are temporary and used to check the value of a variable while debugging the simple methods, it is handy to place an immediately recognizable String at the start of the message to make the message stand out in the logs. For example, place the following log message inside the `removePlanetReview` simple-method in `LearningServices.xml`:

```
<log level="info" message="Removing planetId =
${parameters.reviewId}"/>
```

Then fire an http request `ListPlanetReviews` to our learning webapp and remove a review.

If we have the console log logging to the console and not the file we should see:

```
2008-07-19 11:05:01,828 <http-0.0.0-8443-Processor4> [ ConfigXMLReader.java
:291:INFO ] RequestMap Created: <46> records in 0.0s
2008-07-19 11:05:01,843 <http-0.0.0-8443-Processor4> [ ConfigXMLReader.java
:385:INFO ] ViewMap Created: <31> records in 0.0s
2008-07-19 11:05:01,843 <http-0.0.0-8443-Processor4> [ RequestHandler.java
:236:INFO ] [Processing Request]: RemovePlanetReview sessionId=86CDA709D2E792094
D3FE48BB620E434.jvm1
2008-07-19 11:05:01,906 <http-0.0.0-8443-Processor4> [ UtilXml.java
:243:DEBUG] XML Read 0.063s: file:/C:/workspace/ofbiz4.0/hot-deploy/learning/scr
ipt/org/ofbiz/learning/learning/LearningServices.xml
2008-07-19 11:05:01,906 <http-0.0.0-8443-Processor4> [ Log.java
:94 :INFO ] Removing planetId = 10010
2008-07-19 11:05:01,921 <http-0.0.0-8443-Processor4> [ ServiceDispatcher.java
:467:DEBUG] Sync service [learning/learningRemovePlanetReview] finished in [78]
milliseconds
2008-07-19 11:05:01,921 <http-0.0.0-8443-Processor4> [ RequestHandler.java
:425:INFO ] [RequestHandler.doRequest]: Response is a view. sessionId=86CDA709D2
E792094D3FE48BB620E434.jvm1
```

Notice that our log message is nestled amongst other messages and we had to scroll back in the logs to find it. Now change the log message to:

```
<log level="info" message="***** Removing planetId =
${parameters.reviewId}"/>
```

Remove another review and take a look in the logs:

```
2008-07-19 11:05:01,828 <http-0.0.0-8443-Processor4> [ ConfigXMLReader.java
:291:INFO ] RequestMap Created: <46> records in 0.0s
2008-07-19 11:05:01,843 <http-0.0.0-8443-Processor4> [ ConfigXMLReader.java
:385:INFO ] ViewMap Created: <31> records in 0.0s
2008-07-19 11:05:01,843 <http-0.0.0-8443-Processor4> [ RequestHandler.java
:236:INFO ] [Processing Request]: RemovePlanetReview sessionId=86CDA709D2E792094
D3FE48BB620E434.jvm1
2008-07-19 11:05:01,906 <http-0.0.0-8443-Processor4> [ UtilXml.java
:243:DEBUG] XML Read 0.063s: file:/C:/workspace/ofbiz4.0/hot-deploy/learning/scr
ipt/org/ofbiz/learning/learning/LearningServices.xml
2008-07-19 11:05:01,906 <http-0.0.0-8443-Processor4> [ Log.java
:94 :INFO ] ***** Removing planetId = 10010
2008-07-19 11:05:01,921 <http-0.0.0-8443-Processor4> [ ServiceDispatcher.java
:467:DEBUG] Sync service [learning/learningRemovePlanetReview] finished in [78]
milliseconds
2008-07-19 11:05:01,921 <http-0.0.0-8443-Processor4> [ RequestHandler.java
:425:INFO ] [RequestHandler.doRequest]: Response is a view. sessionId=86CDA709D2
E792094D3FE48BB620E434.jvm1
```

The asterisks make our log message stand out much clearer to the naked eye. Obviously, if we had been printing out to the `console.log`, we could have searched the log file for `planetId` and been taken directly to the correct message.

Debugging Java Code

Eclipse can connect to a remote Virtual Machine (VM) running a Java application and connect it to the internal debugger. Once successfully connected debugging is almost as if we were debugging a local application that is running within the IDE. We can inspect variables, set breakpoints, and step through code, giving us complete visibility as to what is happening with our code at runtime.

The advantage of this is that we can connect and debug through code (providing that the code being executed remotely is identical to our local code) in OFBiz without having to spend time getting it running inside our IDE and can debug the application out of the box. Should we need to, we could connect to another developer's application or even our test staging platform's application and debug. Though this is the OFBiz recommended way for debugging java code, there is also a more traditional solution. You can see how to use it, look for *Debugging (or running) OFBiz in Eclipse* at <http://docs.ofbiz.org/x/gQ>. We do not have to do anything special to OFBiz to get this to work. We just have to make sure that the correct parameters are passed in to the VM at runtime. These parameters have already been added for us, but commented out. All we need to do is make sure that we start OFBiz using the correct command.

Passing in the VM Parameters

Open `startofbiz.bat` and place REM before the first line:

```
"%JAVA_HOME%\bin\java" -Xms256M -Xmx512M -Duser.language=en -jar  
ofbiz.jar > runtime\logs\console.log
```

And delete REM from the last line:

```
"%JAVA_HOME%\bin\java" -Xms256M -Xmx512M -Duser.  
language=en -Xdebug -Xnoagent -Djava.compiler=NONE -  
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005 -jar  
ofbiz.jar > runtime\logs\console.log
```

The next time OFBiz is started the VM is configured to use the **Java Platform Debugger Architecture (JPDA)** and we can connect to our application and debug using port 5005. In the unlikely event that port 5005 is in use by something, simply change this port to a free one.

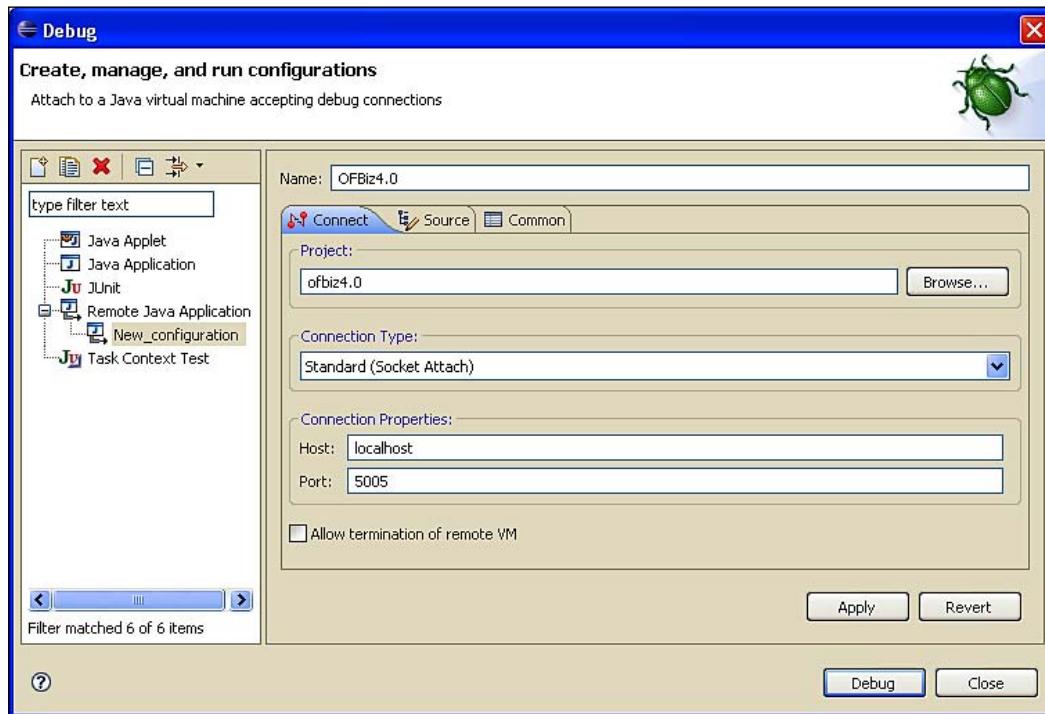
It is important that the VM is not configured to use the JPDA in production. If anyone connects their debugger to this port they can cause the application to hang just by placing a breakpoint in a commonly called piece of code.

Configuring the Remote Debugger

With OFBiz started, open our Eclipse project and click on the down arrow just to the right of the bug icon and select **Open Debug Dialog**:



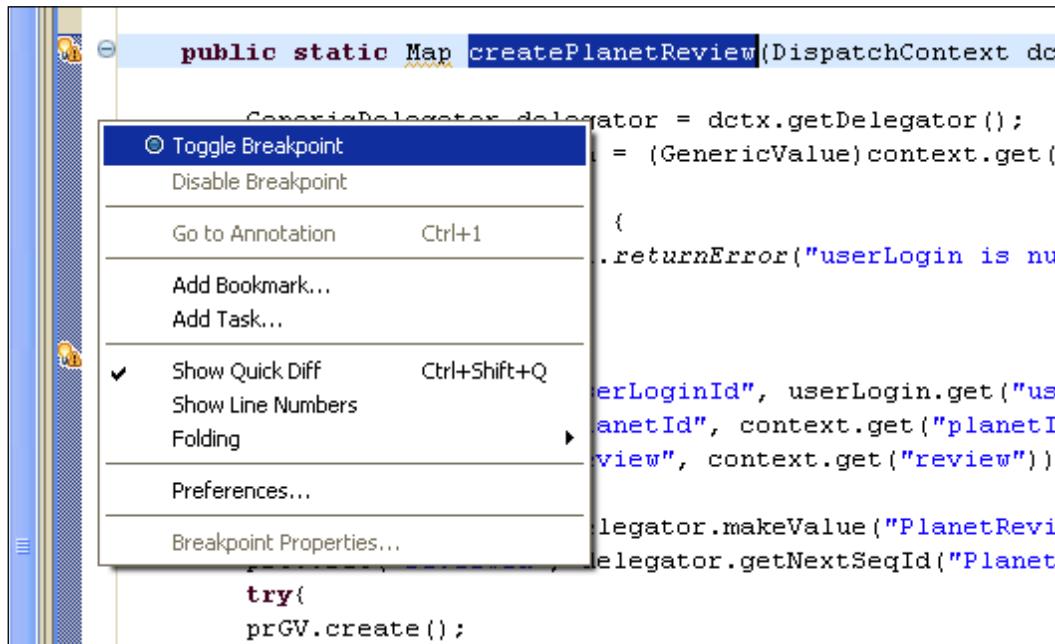
The **Debug** dialog will open. In the left-hand panel double-click on **Remote Java Application** and in the **Name** field type **OFBiz4.0**, in the **Project** field **Browse** for our **ofbiz 4.0** project and in the **Port** field type **5005**, then click **Apply** to save these details.



Tips and Techniques

After the settings are saved, click **Debug** to connect the debugger. In the future to connect the debugger select the arrow to the right of the bug icon and choose **OFBiz4.0**.

Open up the class `org.ofbiz.learning.learning.LearningServices`, find the method `createPlanetReview` and in the left-hand margin of the first line containing code, right-click and select **Toggle Breakpoint**:



When this line of code is executed, the application will pause at this breakpoint and wait for a command like continue or step to the next line. Your local copy of the code must be identical to the compiled code that is being executed or some strange results will be noticed. Pausing on or stepping to an empty line are the most common strange effects.

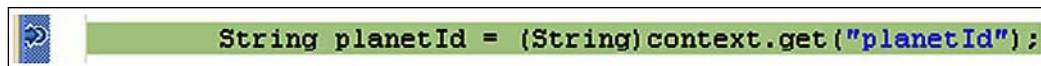
Fire an http request `PlanetReview` to our webapp `learning` and create any review. The application appears to hang, the browser page shouldn't have changed and Eclipse should be flashing Orange in the task bar:



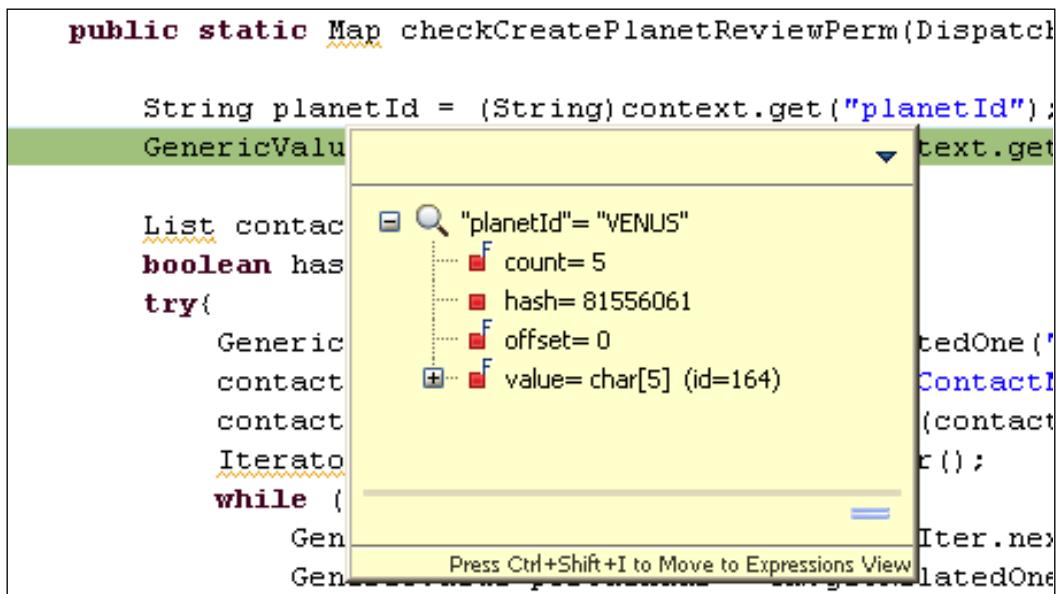
If we return to Eclipse we are now in the **Debug** perspective, and we can see that the thread is suspended in the **Debug** window in the top-left corner.

We can now step through our code, inspecting variables and using Eclipse's full range of debugging tools. Click on the top level **Run** menu to see a full list of possibilities.

We should also notice that the line we set the breakpoint on is now highlighted:



Hit **F6** to step over this line to the next line. Now double-click on the **planetId** variable to highlight it, right-click and select **Inspect** to see information on the value of this variable:



Once finished select the **Disconnect** button from the panel to disconnect the debugger:



Finally select the **Java** perspective in the very top-right-hand corner to return to our original perspective:



Managing a Project Using Subversion

The OFBiz project uses SVN as its version control system. SVN is a very good tool for organizations and developers to use to keep track of the changes to their bespoke code as well. It becomes particularly useful when more than one person is working on the project.

In Chapter 1, we checked out the OFBiz source code using SVN from the Apache OFBiz release 4.0 branch and throughout the book we have worked directly on this code. At any time we could have, through Windows Explorer, right-clicked and updated the code. Tortoise SVN would connect back to the same location and download any changes that have been committed to this branch since our last update or the initial checkout. These changes will be merged with our local changes. If a line or chunk of code has been changed locally, and the same lines have also been changed in the repository, when we update we will get a conflict. These conflicts must be resolved or the code will not function as expected.

So long as the core code does not significantly change locally, and all of the customizations are extracted and placed in the hot-deploy directory, conflicts can be kept to the minimum and should be quick to resolve.

Although this method of working on code directly checked out from the OFBiz repository works and appears to be quite simple, there are serious limitations to working in this way. Local code can only be synced with the repository that it was checked out from. Since we cannot commit back to the OFBiz repository, this fact immediately removes our ability to use any sort of versioning system, other than taking regular backups by copying the code to another location (whether zipped up or not) and time stamping them so we can revert back if something goes wrong. This is not ideal for a number of reasons:

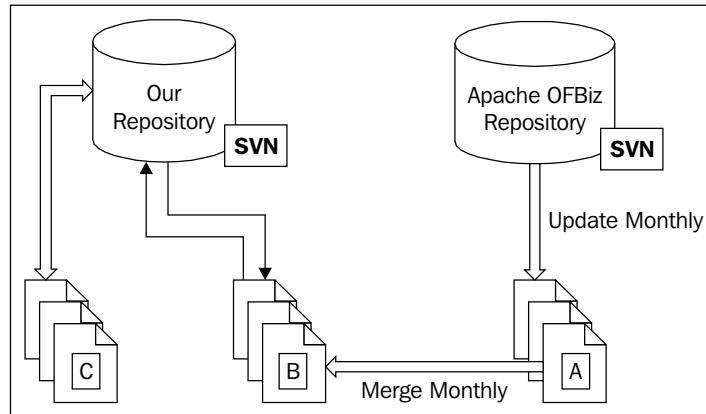
- OFBiz is a large project and the time taken to perform this copying and the space taken to store all the backups makes it not feasible.
- Unless detailed notes are kept regarding which files have been altered, added, or deleted in whichever backup, there is no way of knowing what has changed between backups.
- There is no way for a team to realistically work on the same project in this manner. There is no way for one developer to easily obtain another developer's changes.

A much better solution is to create your own SVN repository at the very start of your project. Even if you are the sole developer at the start of the project, by the end of the project there may be a team. Besides, if you make a mistake, the changes can quickly and easily be rolled back to an earlier revision.

Full details on how to setup your own SVN repository can be found in the Subversion book.

The basics steps of this technique, known as vendor branches, are as follow:

1. A copy of the code is checked out from the Apache OFBiz repository. This code is never altered. (Code A)
2. An exact copy of this code is made locally with the svn hidden files removed, giving us purely the OFBiz code. (Code B)
3. Code B can now be imported to our own repository. Once imported this code should not be changed locally.
4. Developers in our team can now checkout a copy of this code (Code C) and start to commit changes back to the repository.
5. After a given time period (monthly for example), Code A and Code B should both be updated from their respective repositories. Changes made to Code A should be merged into Code B and Code B committed back to our repository. When the developers next update their local copies of Code C, they will have obtained the latest features and bug fixes from the OFBiz repository.



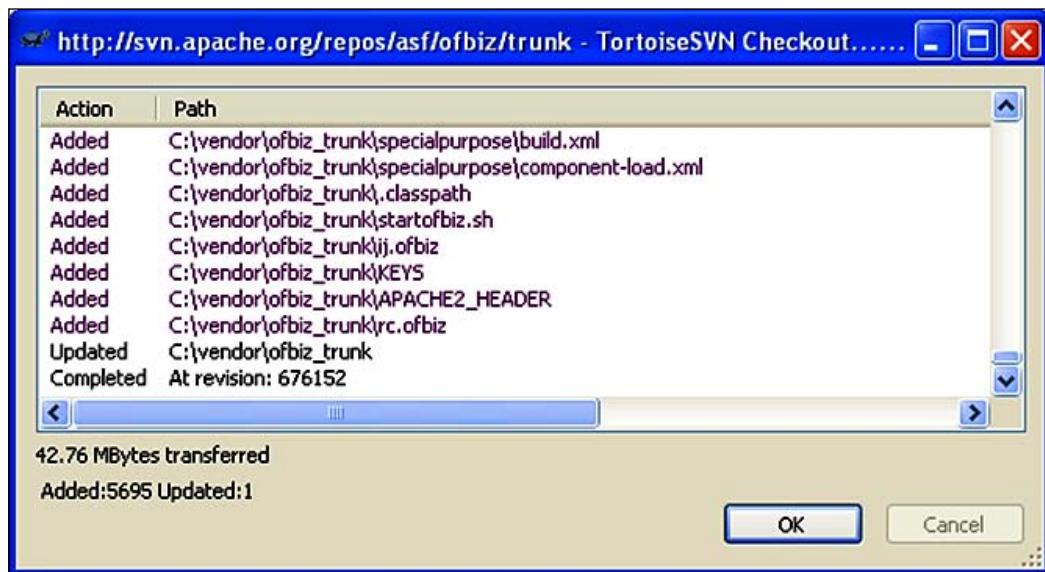
Preparing the Repository and Creating the Project

This section prepares the repository and will ultimately give us Code C to which developers will add their changes.

Step 1: Checking Out the Latest OFBiz Code

To checkout a copy of the latest version of OFBiz, right-click from Windows Explorer and select **SVN Checkout**. As the URL of the repository enter: <http://svn.apache.org/repos/asf/ofbiz/trunk> and for the **Checkout directory** enter `C:\vendor\ofbiz_trunk`.

At the end of the checkout process take a note of the revision number:



In this case the revision number is **676152**.

Step 2: Making a Copy

To get an exact copy of the code we have just downloaded without the hidden SVN files, we can perform an **SVN Export**.

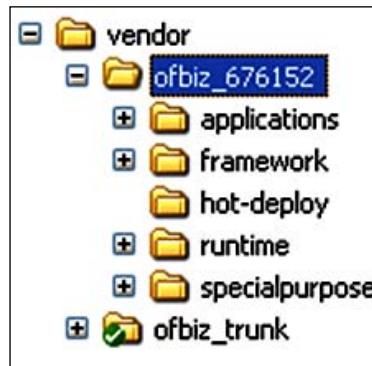
In Windows Explorer create a new directory in our vendor directory called `ofbiz_676152`. Needless to say, the revision number should be changed accordingly.

Right-click on our newly checked out `ofbiz_trunk` directory and select **TortoiseSVN** then **Export**. Once the **Browse For Folder** screen has loaded, find and select the `ofbiz_676152` folder from the list and click **OK**. The files will now be copied across to the new directory. This process will be much faster than the initial checkout since the files are simply being copied from one local directory to another.

Our directory structure should now look like:



Ideally we would like all of the OFBiz files under the `ofbiz_676152` directory and not under an `ofbiz_trunk` sub directory. Simply select all the files in the sub directory and move them up a level. Our directory structure should now look like this:



Step 3: Importing Our Copy

First of all we must create a repository where our project will live. If your organization does not already have an SVN server there are detailed instructions on how to set up a server and also create a repository in the SVN book (<http://svnbook.red-bean.com>).

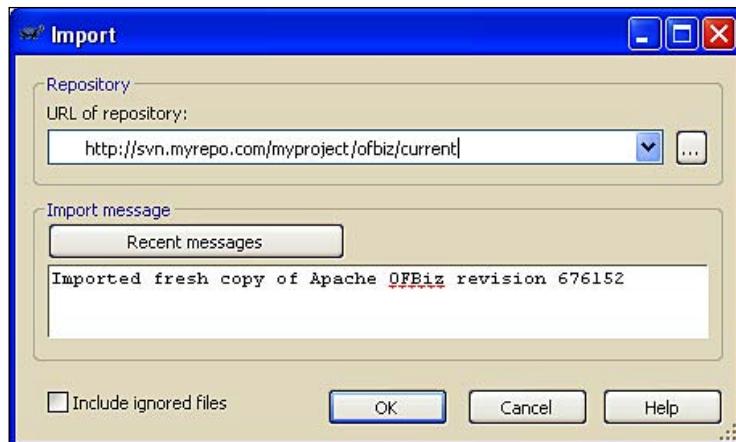
We will assume that a repository called `myproject` has been created and secured so that we have access to read and write to it.

Right-click on the `ofbiz_676152` folder and select **Repo-Browser** and enter the URL of the newly created repository. Once browsing the empty repository create a new folder structure by right-clicking and selecting **Create Folder**. The repository structure should look like this:

File	Extension	Revision	Author	Size	Date
http://svn.myrepo.com/myproject					
ofbiz		1	rhowell		19/07/2008 14:20:50
current		1	rhowell		19/07/2008 14:20:50
vendor		1	rhowell		19/07/2008 14:21:24

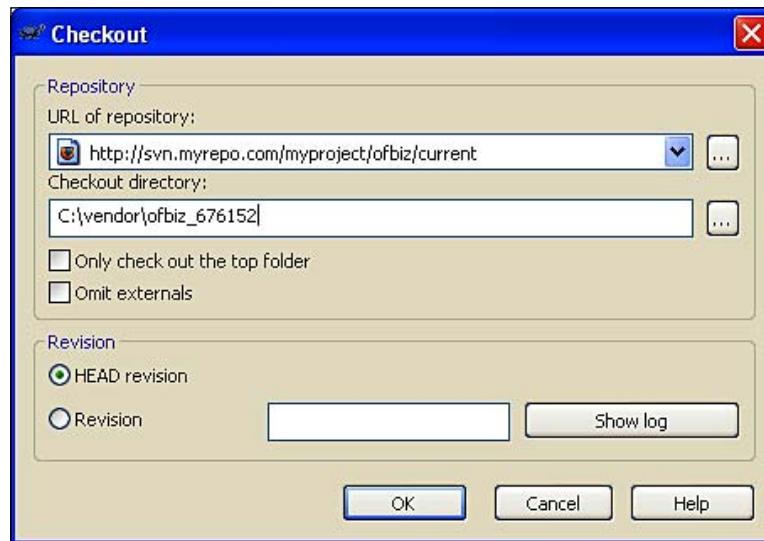
Tips and Techniques

Finally, we are ready to perform the import. Our copy of revision **676152** of the Apache repository is going to be imported into the current directory in our repository. In Windows Explorer right-click on the folder `ofbiz_676152` and select **TortoiseSVN** and then **Import**:



Select **OK** to perform the import. The import may take some time depending on network or connection speed. However, once this is complete, we will have a clean copy of the OFBiz code committed to our repository.

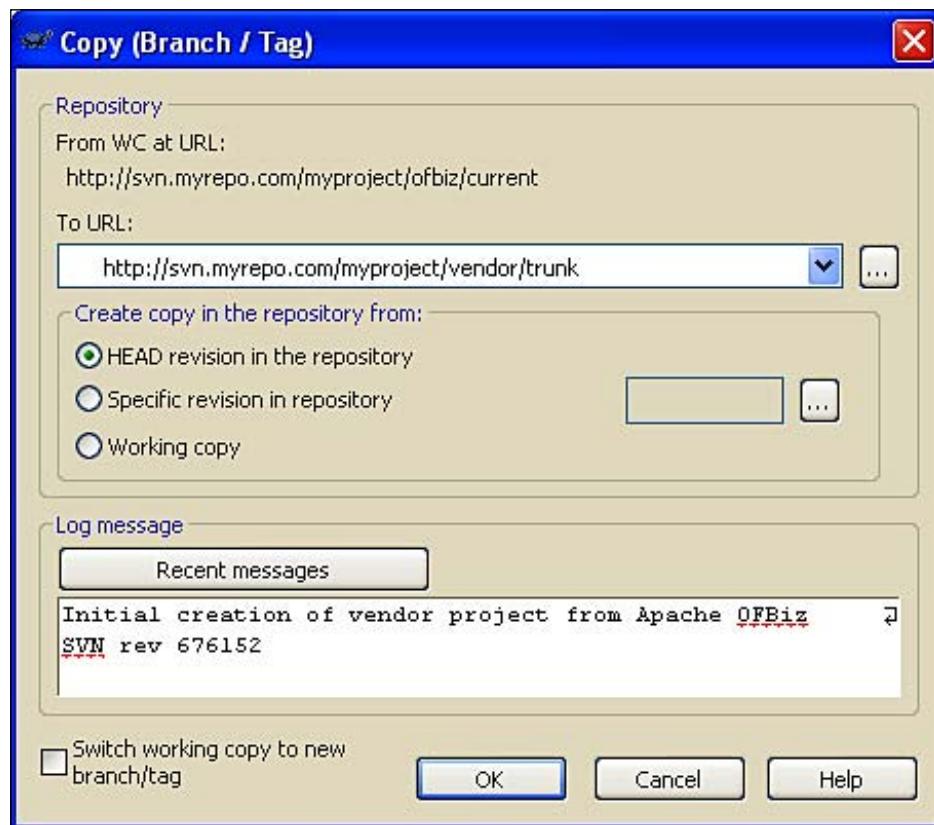
There is an SVN oddity in that the code we have just imported is not under version control. Once the import has finished delete the local copy of `ofbiz_676152` and re-check it out back to a folder of same name:



Step 4: Branching ofbiz\current to Create the Trunk

We want to keep ofbiz\current as it was when we checked it out from the Apache SVN server so we have a copy of the code base from which we work on. We want the developers to work on a branch of OFBiz current.

To create the branch, right-click on the freshly checked out local ofbiz_676152 folder and select **TortoiseSVN | Branch/tag** and change the **To URL** field to <http://svn.myrepo.com/myproject/vendor/trunk>.



This is the URL to the SVN that can now be shared by the developers. This is your bespoke project and will contain all of the customizations. Once the code is checked out from this location this is Code C of the earlier diagram.

This may have seemed like a drawn out process to create the repository, however it is important that the repository is well structured, and that the structure and reason for the structure is understood from the beginning of the project so that every change can be tracked and merges performed with relative ease.

Getting the Latest Features and Bug Fixes

And so the developers have got to work creating bespoke components and customizing existing ones, adding new functionality and changing the design! A month has passed and 200 commits have made back to the vendor\trunk branch. We've been monitoring the OFBiz development mailing lists and have been getting daily updates on the commits that have been made to OFBiz and tracking the issues in JIRA and we know that some important bugs have been fixed and a piece of functionality we've been waiting for is finished and is looking stable.

It's time to get these changes and merge them back into our project.

Step 5: Updating OFBiz Current

Return the local folder in *Step 2* and update the ofbiz_trunk folder. This will bring our local copy up to the same revision number as the Apache OFBiz SVN repository. Keep an eye out for the revision number in the update window but if you miss it, simply right click on ofbiz_trunk in Windows Explorer and select **Properties**, then click the **Subversion** tab to get the revision number.

If this original code has been lost or deleted do not worry, simply perform a fresh checkout from the OFBiz repository. Follow *Step 2* through until we have a new local folder:

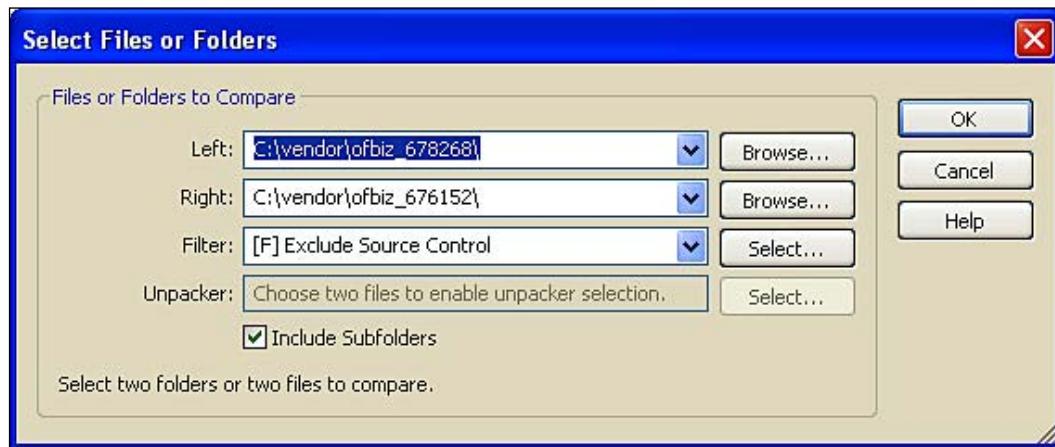


We are going to perform two merges. The first merge is simple. We are going to merge the changes in ofbiz_678268 into ofbiz_676152 and commit the latter back into the repository (don't forget that this will be committed back into the ofbiz\current branch of the repository).

For the second merge, we are going to get SVN to work out the changes between ofbiz_678268 and ofbiz_676152 and merge these changes locally into the latest version of our custom project. We can then commit these changes back into vendor\myproject.

To perform the first merge, we are going to use WinMerge to give us maximum visibility as to what is happening. WinMerge is an Open Source visual test file differencing and merging tool for Windows and will graphically display the differences. Download and install the latest version from the WinMerge website at <http://winmerge.org>.

Open WinMerge and from the top menu select **File** then **Open**. The latest revision is going to be **Left** and the older revision **Right** and add the Filter **Exclude Source Control**. Finally ensure that **Include Subfolders** is selected:



Click **OK** to return a complete list of every file that has been compared with the results of the comparison. Click the **Comparison Result** column header and sort (descending) by this column.

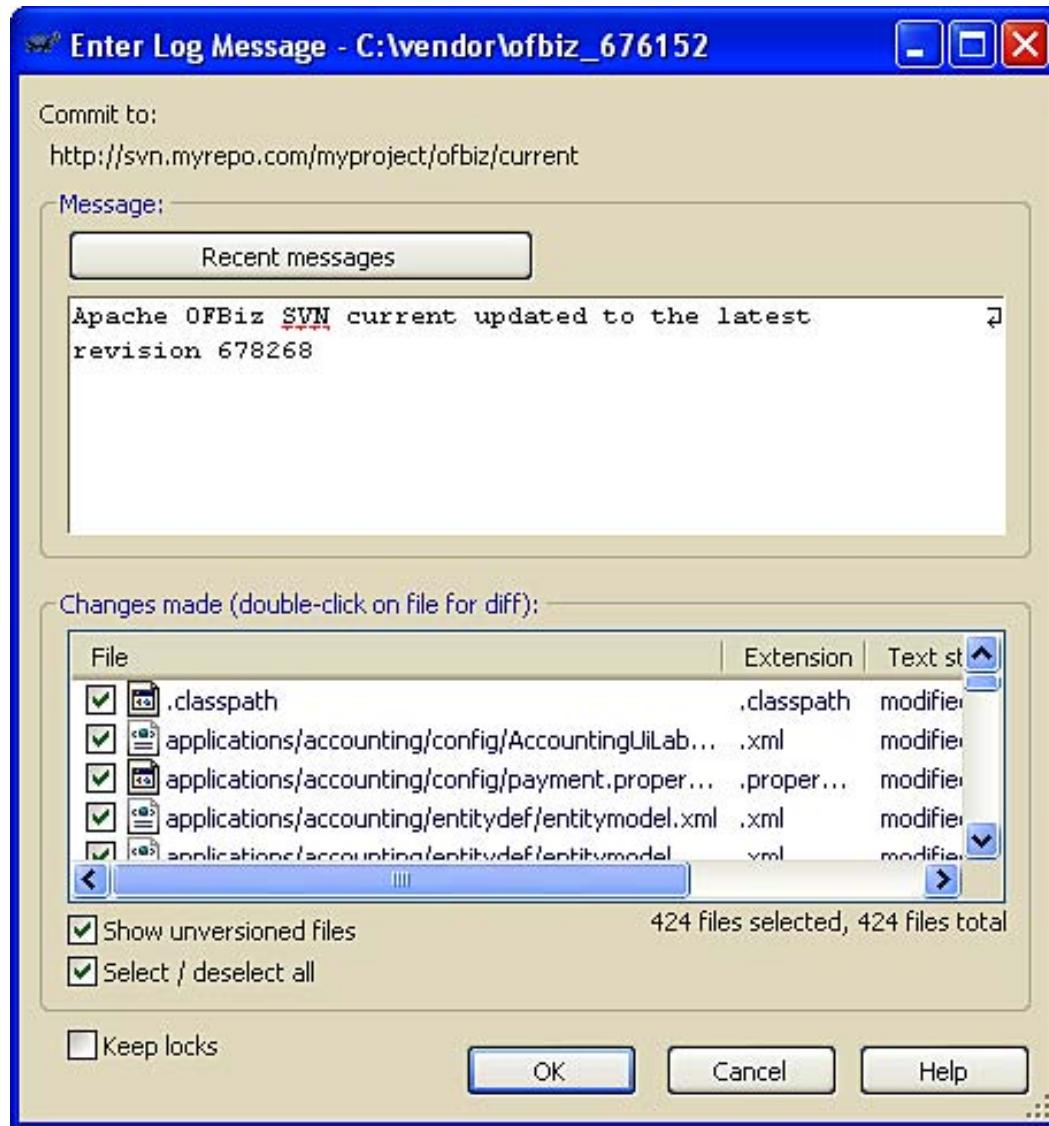
Left Only means that the files appear in the later revision and not in the earlier revision. They are new additions and need to be copied from left to right. Carefully select all of these files and click the **Copy Right** icon.

Right Only means that the file exists in the early revision and not in the later version. They have been deleted and must be deleted from right. To do this, carefully select all of these files and folders and right-click and select **delete**.

Sort once again by **Comparison Result** and select all of the **Files are Different** and **Binary Files Are Different** and **Copy Right**.

Tips and Techniques

Return to Windows Explorer and commit the `ofbiz_676152` folder back into the repository with a message: **Apache OFBiz SVN current updated to the latest revision 678268**. In the commit window click the **Select/deselect all** checkbox twice to ensure that all unversioned and removed files are going to be added to or deleted from the repository.



Once committed, delete the `ofbiz_678268` directory and rename the `ofbiz_676152` directory to `ofbiz_678268`.

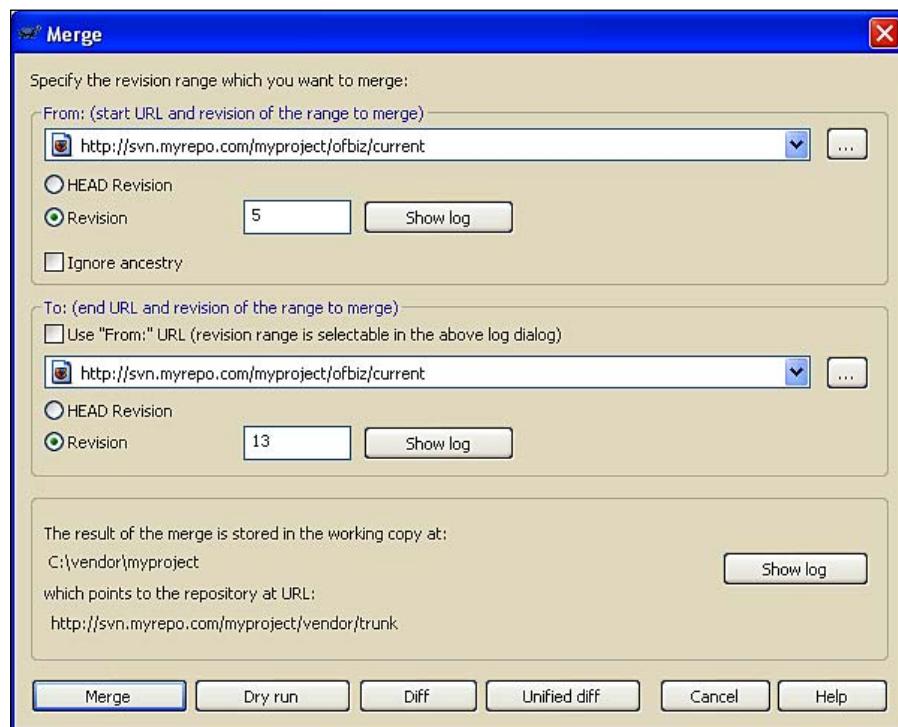
We are now maintaining an up-to-date, clean version of OFBiz in our repository, that increments in line with our merging strategy. One revision in the **ofbiz\current** branch could encompass thousands of revisions in the Apache SVN repository. By doing this we can keep a reference to exactly what was merged with our project and when.

We are going to use TortoiseSVN to merge the changes made between the two revisions of **ofbiz\current** into our local copy of the **vendor\branch**.

Step 6: Merging Changes between Revisions of OFBiz Current into Our Project

Right-click on the **ofbiz_678268** folder and select **TortoiseSVN | Show Log** to determine the revisions that the **ofbiz\current** branch has been through. In this case **ofbiz\current** was created and the initial import was performed in revision 5 and the last commit, where it was brought up to revision 678268, was revision 13.

Right-click on a newly updated local version of the custom project. This local code should be identical to the repository's **vendor\trunk** and should not contain any local changes. Select **TortoiseSVN and Merge**:



Although no changes are made to the repository and all changes can be reviewed and reverted locally before being committed, it may be wise to perform a **Dry Run** first to see the changes that are going to be made.

Once the merge has been completed we are left with a number of locally modified, deleted, and conflicted files. These changes should be reviewed and conflicts resolved. This task should be undertaken with great care, and it may be easier to perform using Eclipse and the Subclipse plugin. Once completed the project should be re-built and tested. Once satisfied that the merge has been successful, it should be committed back to the vendor\trunk with an appropriate comment

Apache HTTP Server and OFBiz using mod_proxy_ajp

Although OFBiz runs out of the box and does not necessarily require an http server to handle the requests, many people prefer to run OFBiz behind the Apache HTTP Server to give them extra functionality, simplicity and scalability. SSL certificates are easier to manage with the Apache HTTP Server. Apache's HTTP Server, abstracts the backend architecture making it easier to create a scalable, multi-server, load balanced architecture.

Installing the Apache HTTP Server

Download and install the latest version of the Apache HTTP Server from the Apache website <http://httpd.apache.org/download.cgi>. It is preferable to download the version with mod_ss1 included. At the time of writing the best available version was 2.2.9 Win32 Binary including OpenSSL 0.9.8h (MSI Installer).

Follow the on-screen instructions to install the server. If you are installing on your local machine, the default values throughout the installation can be used.

Before we make any changes to any configuration files we need to make sure that the **http server can be started and stopped without any errors**. One likely error on start-up is the could not bind to address 0.0.0.0:80 . This error means that another application is using one of the ports the http server wants to use and can be usually rectified by shutting down the other application (on Windows if you are running **Internet Information Server (IIS)** this must first be shutdown). More information on installation issues is available at the Apache HTTP Server website: <http://httpd.apache.org>.

To see errors appear in a console window, start the server by selecting **Start | Programs | Apache HTTP Server 2.2 | Control Apache Server | Start**.

Once started you see the icon below in the task bar in the bottom-right of the screen next to or near the clock. By default, Apache attempts to start on completing installation.



What is mod_proxy_ajp?

`mod_proxy_ajp` is a support module for the AJP (Apache JServ Protocol) 1.3 allowing the web server to communicate with the servlet container over TCP connections. In this case, the web server is the Apache HTTP Server we have just downloaded and installed and the servlet container can be thought of as our Apache OFBiz instance. Really, the servlet container it is connecting to is actually the embedded Tomcat servlet container that serves our application. By loading and configuring the connector in the Apache HTTP Server, and telling OFBiz which port to listen for the connector on, we can proxy requests through the http server to our application.

Configuring Apache HTTP Server to Use mod_proxy_ajp

These days Apache ships with everything we need to use the AJP connector, however by default, the modules are not loaded.

Open Windows Explorer and navigate to the conf folder inside the Apache install folder (by default this is C:\Program Files\Apache Software Foundation\Apache2.2\conf).

Open `httpd.conf` in a text editor and scroll down to the section where the modules are loaded using the `LoadModule` directive. A # sign preceding a line means that this line is commented out. There are three modules which must be loaded so find the following lines and uncomment them by deleting the #.

```
#LoadModule proxy_module modules/mod_proxy.so  
#LoadModule proxy_ajp_module modules/mod_proxy_ajp.so  
#LoadModule ssl_module modules/mod_ssl.so
```

To keep our changes to the HTTP server's main configuration file to an absolute minimum we can include any number of configuration files. The main bulk of changes can go into these smaller configuration files. Head to the very end of the `httpd.conf` file and find and uncomment the following line:

```
#Include conf/extra/httpd-vhosts.conf
```

Open this `httpd-vhosts.conf` file and immediately above the `NameVirtualHost` `*:80` directive add:

```
ProxyRequests Off
<Proxy *>
    AddDefaultCharset off
    Order deny,allow
    Allow from all
</Proxy>

ProxyVia On
```

Next inside both the `*.80` and `*.443` `VirtualHost` directives set the `ServerName` directive to `localhost` add the following lines, just before the closing tag:

```
ProxyRequests Off
ProxyPreserveHost On
ProxyPass / ajp://127.0.0.1:8009/
```

Configuring OFBiz to Use the AJP Connector

Like so many other things in OFBiz, this has already been done for us so very little configuration is needed.

Open the `ofbiz-containers.xml` file in `framework\base\config` and take a look at the `<container>` element named `catalina-container` and scroll down to:

```
<property name="ajp-connector" value="connector">
```

All properties needed for the connector are defined here. Most importantly look for the **port** property and notice that this tallies with the port specified in the `ProxyPass` directive.

```
<property name="port" value="8009"/>
```

To change the port the connector uses both locations must be changed in both Apache and OFBiz.

It is likely that in production, ports 8080 and 8443 will be firewalled. If we are using the `ajp-connector` it is possible to remove the `http-connector` and the `https-connector` by commenting out the entire `<property>` tags. They are no longer needed. Be careful that you remove the existing comments first since you will get an error as XML does not allow nested comments.

While we are in this file, it is also wise to comment out the `<beanshell-container>` as well. This container is a serious security risk. It opens two ports for allowing telnet access directly into the JVM. These ports should be protected by a firewall. If this container is loaded and the ports are insecure, it would be possible to execute BeanShell code on your instance of OFBiz. This could be disastrous.

Getting the Secure Pages to Work

With Apache HTTP server already running, start OFBiz and then open a browser. Now navigate to `http://localhost/ecommerce` (notice we are no longer using port 8080 – our requests are now being proxied by Apache and we direct them to port 80 as defined by the `VirtualHost` directive). The page should load as normal and all of the links in the pages should now correctly point to `http://localhost/ecommerce/control/****`. Now select the **Login** link. The page that it is redirected to is still 8443. As the `https-connector` has been disabled in the `ofbiz-containers.xml` file, this page will not load.

Open the file `url.properties` in `${ofbizInstallFolder}\framework\webapp\config` and change the properties:

```
port.https=8443  
port.http=8080
```

to:

```
port.https=443  
port.http=80
```

Changing the port settings alone is not enough for the secure pages to work. All that this does is change the links that are transformed by the `<@ofbizUrl>` and `<@ofbizContentUrl>` tags. Browsers will not load pages on port 443 without an SSL certificate of some kind. For the sake of the example we will be creating a self-signed certificate. For testing purposes this will be acceptable (although we will see a warning when we first enter the site). For production purposes a valid certified SSL certificate should be in place of the self signed one.

Configuring SSL in Apache

Once again open the http server's main configuration file, the `httpd.conf`, located in the `conf` directory of its install directory.

Towards the bottom of the file, find and uncomment the following line:

```
#Include conf/extra/httpd-ssl.conf
```

Inside the `httpd-ssl.conf` file change the `VirtualHost` Directive from

```
<VirtualHost _default_:443>
```

to

```
<VirtualHost *:443>
```

Next change the `ServerName` to `localhost:443` and finally to bottom of the file, just above the closing `</VirtualHost>` tag, add the following just as we did to the `VirtualHost *:80` directive block:

```
ProxyRequests Off  
ProxyPreserveHost On  
ProxyPass / ajp://127.0.0.1:8009/
```

Creating a Self-Signed SSL Certificate

Full instructions for creating SSL certificates using OpenSSL which is bundled with the version of Apache we have installed can be found on the Apache website at http://httpd.apache.org/docs/2.2/ssl/ssl_faq.htm. To create a self-signed SSL Certificate for testing purposes simply open a Command Prompt and change directory to the bin directory. If the default location was chosen at installation this will be: `C:\Program Files\Apache Software Foundation\Apache2.2\bin` and enter:

```
openssl req -new -x509 -nodes -config ..../conf/openssl.cnf -out ..../conf/server.crt -keyout ..../conf/server.key
```

Now answer the questions. The response to **Common Name (CN)** should match the server name. In this case it is `localhost`.

This will create two files: `server.crt` and `server.key` in the `Apache2.2\conf` directory.

Restart the Apache HTTP server and ensure that OFBiz is still running, then navigate the browser to `http://localhost/ecommerce` and attempt to log in. You will be once again warned that the **Website is Certified by an Unknown Authority**. Accept this certificate permanently and the secure OFBiz pages will now load.

Going into Production

This is a basic guide to help get you off the ground running Apache and OFBiz locally on the same machine. Had we wished to put OFBiz onto a different machine on our network we could have replaced the `ProxyPass` directive to point to the IP Address of the machine on which OFBiz is running. For example:

```
ProxyPass / ajp://10.10.10.100:8009/
```

Before going into production using this architecture, an intricate knowledge of your network and also of the HTTP server should be gained. There is a lot more information on the HTTP server on both the Apache website and also hundreds of other forums and help sites across the web. It is after all the world's most commonly used web server.

Development Tips

As large a project as OFBiz is, once the basic rules, patterns, and concepts are understood even the most complex functionality can be broken down into simpler parts. There are a number of simple tips that will aid the development of custom ERP solutions. Many may seem obvious, and all of them apply to any Open Source software project.

Familiarize yourself with What is there

Before attempting any customizations, a full understanding of all of the functionality that is present in OFBiz should be gained. OFBiz is a full suite of ERP solutions. These solutions are other developer's answers to questions and problems that their businesses have faced in the past. More often than not, solutions that are needed by your business are already in place or similar solutions can be changed to meet your requirements.

Reuse What is there

Re-inventing the wheel is time consuming and will often leave you with a product that is not as good as the original! Wherever possible, reuse what is there. The collective experience of many developers and businesses can often be greater than your own. Unless you have familiarized yourself with what is already there, you will not be able to reuse code to its maximum potential.

Follow the Best Practices

The best practices guide laid down by the OFBiz committee at <http://docs.ofbiz.org/display/OFBADMIN/OFBiz+Contributors+Best+Practices> should be adhered to from the start of the project. Sticking to these best practices and coding conventions will make it easier to upgrade, or should you wish to, contribute back to the project.

Read the Documentation and Books

The OFBiz website is the place to start. The confluence documentation site is growing and now contains a large repository of must read material. There are number of books recommended on the OFBiz site, some will be more useful to your business needs than others. The *Data Model Resource* book by **Len Silverstone** will give the greatest knowledge of the underlying data model.

Subscribe to the Mailing Lists and become an Active Member

Mails from the mailing lists can be received individually as they are sent once or twice a day in digest form. They are a fantastic tool to aid learning and discovering new functionality that is being released. Mails are answered promptly, but remember that they are answered by other developers who are not getting paid to respond. If you are working closely on an OFBiz project you will start to see questions that you know the answer to because you have struggled with and solved for yourself. If you share this knowledge, when you next have a problem, maybe they will return the favor.

Develop on a Fast Machine

This is particularly important when developing large applications like OFBiz. A powerful machine will compile OFBiz and build the project in seconds. A machine with a lower specification could take minutes. When programming Java events and services, where re-builds and restarts are constantly necessary, having a faster machine will significantly decrease development time and will not test the developer's patience nearly as much.

Contribute

Contributing features and fixes back to the OFBiz project will naturally keep your project closely aligned to the OFBiz trunk. Contributions help the project to grow and the closer your projects are aligned, the easier it is for your project to be merged and upgraded.

Summary

In this chapter we have looked at some techniques to help development and some steps to take before development starts to ensure we begin with a solid footing. We have learned more about configuring and outputting to the OFBiz logs and have looked at how to debug OFBiz using the Eclipse remote debugger to step through the Java code and inspect variables.

We have examined how we can set up a merging strategy using SVN to ensure that a customer project can be developed and still have the latest bug fixes and functionality added to it from the OFBiz project.

We investigated an architectural design and saw how easy it is to connect the Apache HTTP Server to OFBiz using the Tomcat connector `mod_proxy_ajp`.

In this chapter, we looked at:

- Debugging Techniques
- Maintaining a project with the latest bug fixes and features
- Working with the latest version
- Running OFBiz with Apache HTTP Server using `mod_proxy_ajp` connector
- Development Tips

Simple Method User's Guide

The Simple Methods user guide specifies some of the most commonly used elements and attributes. Although more are available to use, they have been omitted from this guide as they are either deprecated, or simply not used. The complete XML schema from which this guide has been derived can be found at \${ofbizInstallFolder}/framework/minilang/dtd/simple-methods.xsd.

<simple-methods>

Opening tag of Simple Methods files.

<simple-method>

This is the opening tag for the Simple Method. The Simple Method can be called in either an event context from the Control Servlet (or another event), a service context through the Service Engine, or through any other component that has access to a service dispatcher.

Attribute Information:

method-name="..."	Required
The name as preferably a legal Java method name for this method. This name must be unique for the XML file it is in, as it will be used to reference this method externally.	
short-description="..."	Required

A short description briefly describing the function of the method.

Login-required="true|false" (Optional – Defaults to true)

Checks if a user (UserLogin object, or login.username and login.password Strings) is required to be logged to run this method.

Use-transaction="true|false" (Optional – Defaults to true)

Determines whether the framework should create a transaction if none exists for the thread.

Usage

```
<simple-method method-name="createPostalAddressBoundary" short-
description="Create Postal Address Boundary">
...
</simple-method>
```

Call Operations

We are able to call and process the results of static Java methods, BeanShell script and even invoke other services from our Simple Methods. These operations are available when the static method is called as an event or a service.

<call-map-processor>

The `<call-map-processor>` tag invokes a simple map processor from an existing Map, creating a new Map or adding to an existing one if the name `out-Map` already exists. Resulting messages are added to the name `List`, and a new `List` is created if a `List` with the given name does not yet exist. Note that all `Lists` and `Maps` exist in the same context and must have unique names. An inline `simple-map-processor` can be used by putting a `simple-map-processor` tag under the `call-map-processor` tag. If both, an external and an inline `map-processor` are specified, the external one will be called first, allowing the inline one to override its behavior.

Attribute Information

`xml-resource="..."` Required only if external map processor is desired

The full path and filename on the classpath of the XML file which contains an external map processor to execute.

`processor-name="..."` Required only if external map processor is desired

The name of the external map processor to execute in the specified `xml-resource`.

in-map-name="..."	Required
The name of a map in the method environment to use as the input map.	
out-map-name="..."	Required
The name of a map in the method environment to use as the output map. Will be created if it does not exist already. If it already exists, it will be added to in place.	

Usage

```
<call-map-processor
    xml-resource="org/ofbiz/accounting/payment/PaymentMapProcs.xml"
    processor-name="deletePaymentMethod" in-map-name="parameters"
    out-map-name="context"/>
```

<call-service> / <call-service-asynch>

The `<call-service>` tag invokes a service through the Service Engine. If the specified error code is returned from the service, the event is aborted and the transaction in the current thread is rolled back. Otherwise, the remaining operations are invoked.

The `<result-to-request>` and `<result-to-session>` elements will be ignored when called in a service context. So, they are only used when called in an event context.

The `<call-service-asynch>` tag calls a service asynchronously and ignores the result, so no return messages are used. This doesn't mean no errors will result, but they would just be system errors like database failures, and so on, which all have system error messages.

Attribute Information

service-name="..."	Required
Name of the service to call.	
in-map-name="..."	Required
Optional name of a map in the method environment to use as the input map. If you're not going to pass any parameters to the service then you can just leave off the in-map name, although typically in a service tag you will see a service-name and the in-map name passed in.	
include-user-login="true false"	Optional – defaults to true
by default, will include the user login so if there is a user login for the current simple-method it will pass that in to the service. If you don't want it to pass that in, you can just set this to false.	

break-on-error="true false"	Optional – defaults to true. <call-service> only
-------------------------------	---

If there's an error in the service, by default it will stop the current simple-method and return an error message that came from the service it called. If you don't want it to when there's an error, it can just be set to `false`.

Usage

```
<call-service service-name="updateCreditCard" in-map-name="uccMap"
              break-on-error="false"/>
```

<set-service-fields>

Looks at all of the incoming service attributes as specified in the service definition, looks for fields with the same name in the incoming map and copies those onto the outgoing map.

Attribute Information

service-name="..."	Required
--------------------	----------

Name of the service from which to take all incoming attributes.

map-name="..."	Required
----------------	----------

Incoming map to copy fields from.

To-map-name="..."	Required
-------------------	----------

Map to copy fields to.

Usage

```
<set-service-fields service-name="updateCreditCard"
                     map-name="creditCard" to-map-name="uccMap"/>
```

Handling the Results of Service Calls

The following operations are designed to handle the results, or OUT parameters from a synchronously called service.

<results-to-map>

The <results-to-map> tag takes all of the results of the service, the outgoing maps from the service and puts them in a map of the given map-name.

Attribute Details

<code>map-name="..."</code>	Required
-----------------------------	----------

Name of the map where results will be put in.

Usage

```
<call-service service-name="calculateAcctgTransTrialBalance"
              in-map-name="trialBalanceCallMap">
    <results-to-map map-name="trialBalanceResultMap"/>
</call-service>
```

<result-to-field>

Specify the name of the field in the result and then the name of the field in the context you want to put it in, and optionally the name in the map. If no field-name is specified then the result-name will be used for the field-name, that's the name of the variable that will be created in the current context for the value of that result.

Attribute Details

<code>result-name="..."</code>	Required
--------------------------------	----------

Name of the result. May be used for the field-name. If you don't specify a field-name, this is the name of the variable that will be created in the current context for the value of that result.

<code>field-name="..."</code>	Optional
-------------------------------	----------

Optional field name.

<code>map-name="..."</code>	Optional
-----------------------------	----------

Optional map name.

Usage

```
<call-service service-name="createAgreement"
              in-map-name="createAgreementInMap">
    <result-to-field result-name="agreementId"
                     field-name="agreementIdTo"/>
</call-service>
```

<result-to-request>

The `<result-to-request>` tag is event specific. It takes the result with the given name and puts it in a request attribute with the given name here. The `request-name` is optional. If this is left off then it will put it in an attribute with the name of the `result-name`.

Attribute Details

result-name="..."	Required
-------------------	----------

Name of the result. May be used for the request attribute name. If you don't specify a `request-name`, that's the name of the request attribute that will be created for the value of that result.

request-name="..."	Optional
--------------------	----------

Optional request name.

Usage

```
<call-service service-name="updateCreditCard" in-map-name="context">
<result-to-request result-name="paymentMethodId"
                    request-name="paymentMethodId"/>
</call-service>
```

<result-to-result>

Service specific (when the Simple Method is being called as a service). It will take the result of the service you're calling with the `call-service` operation and it will put it in with the result of the current service. So `result-name` is the name of the result in the service that was called using the `<call-service>` tag.

These attributes are somewhat confusing: `result-name` is the name of the field in the result of this service call that the value comes **from**; `service-result-name` is the name of the field in the result of this simple-method called as a service where the value goes **to**; in other words **from** `result-name` and **to** `service-result-name`.

Attribute Details

result-name="..."	Required
-------------------	----------

Name of the field in the result of this service call that the value comes **from**.

service-result- name="..."	Optional
-------------------------------	----------

Name of the field in the result of this simple-method called as a service where the value goes **to**.

Usage

```
<call-service service-name="createPhysicalInventory"
              in-map-name="createPhysicalInventoryMap">
<result-to-field result-name="physicalInventoryId"
                  field-name="physicalInventoryId"
                  map-name="parameters"/>
```

```
<result-to-result result-name="physicalInventoryId" service-result-
name="physicalInventoryId"/>
</call-service>
```

<call-bsh>

Runs an external bsh script from the classpath if the resource is specified and then runs the inlined bsh script if any is specified. The bsh context is the current simple-method environment including maps, lists and special objects whose names are defined in the simple-method attributes.

The current `env` cannot be modified, but if a Map is returned by the bsh block the entries in the map will be put into the current `env`.

Error messages go on the error list and are handled with the `<check-errors>` tag.

Attribute Details

<code>resource="..."</code>	Optional Name of a properties file on the classpath.
<code>error-list-name="..."</code>	Optional – defaults to <code>error_list</code> The name of the list in the method environment to check for error messages.

Usage

```
<call-bsh><! [CDATA [
    String password = (String) userLoginContext.get("currentPassword");
    String confirmPassword = (String)
        userLoginContext.get("currentPasswordVerify");
    String passwordHint = (String) userLoginContext.get("passwordHint");
    org.ofbiz.common.login.LoginServices.checkNewPassword(newUserLogin,
        null, password, confirmPassword, passwordHint, error_list, true,
        locale);
]]></call-bsh>
```

<call-simple-method>

The `<call-simple-method>` tag calls another simple-method in the same context as the current one. In other words the called simple-method will have the same environment as the calling simple-method, including all environment fields, and either the event or service objects that the calling simple-method was called with.

Attribute Details

`xml-resource="..."` Optional

The full path and filename on the classpath of the XML file which contains an external simple-method to execute. This is only required if a simple-method in a different file is desired.

`method-name="..."` Required

The name of the simple-method to execute in the specified xml-resource, or in the current XML file if no xml-resource is specified.

Usage

```
<call-simple-method method-name="inlineCheckContactListMechType"/>
```

<call-object-method>

Calls a method on an existing object that exists in a field in the environment or in a map in the environment. The string and field sub-elements are passed to the method as arguments in the order they are specified. If the sub-elements do not match the method arguments, an error will be returned. The return value will be put in the named field if a value is returned and if a field and (optionally) Map name are specified.

Attribute Details

`obj-field-name="..."` Required

The name of the field the object is in that has the method to be called.

`obj-map-name="..."` Required

The name of the map the field of the object is in, that has the method to be called. If this is not specified, the field will be taken from the environment.

`Method-name="..."` Required

The name of the method to call on the given object.

`ret-field-name="..."` Optional

The name of the field to put the result in. If not specified, any return value will be ignored.

`ret-map-name="..."` Optional

The name of the map the field of the return value is in. If not specified, but the field name is, then the environment will be used to find the field in.

Usage

```
<call-object-method obj-field-name="USERNAME"
                    obj-map-name="parameters"
                    method-name="toLowerCase"
                    ret-field-name="USERNAME"
                    ret-map-name="parameters"/>
```

<call-class-method>

Calls a static method on a class. The string and field sub-elements are passed to the method as arguments in the order they are specified. If the sub-elements do not match the method arguments, an error will be returned. The return value will be put in the named field if a value is returned and if a field and optionally a Map name are specified.

Attribute Details

class-name="..." Required
The name of the class to call the static method on.
method-name="..." Required
The name of the static method to call on the given class.
ret-field-name="..." Optional
The name of the field to put the result in. If not specified, any return value will be ignored.
ret-map-name="..." Optional
The name of the map the field of the returned value is in. If not specified, but the field name is, then the environment will be used to find the field in.

Usage

```
<call-class-method
    class-name="org.ofbiz.product.product.KeywordSearch"
    method-name="induceKeywords">
    <field field-name="productInstance"
        type="org.ofbiz.entity.GenericValue"/>
</call-class-method>
```

Service Operations

The following operations are service specific, that is, they are only applicable when the Simple Method is called as a service.

<field-to-result>

The <field-to-result> tag copies a field from a Map to the specified service result field.

The tag is only used when the simple-method is called as a service, it is ignored otherwise.

Attribute Details

Field-name="..."	Required
The name (key) of the map field to use.	
map-name="..."	Optional
The name of the map in the method environment. If not specified the field-name will be used to get the field from the method environment.	
result-name="..."	Optional
The name of the request attribute to use. Defaults to the field-name.	

Usage

```
<field-to-result field-name="agreementId" result-name="agreementId"/>
```

Event Operations

The following operations are event specific, that is, they are only applicable when the Simple Method is called as an event.

<session-to-field>

The <session-to-field> tag copies an attribute from the servlet session to a field of a map in the method environment.

Attribute Details

Field-name="..."	Required
The name (key) of the map field to use.	

Usage

```
<session-to-field field-name="visit"/>
```

Environment Operations

The following environment operations deal with moving values from one field to another.

<set>

Moves a value from one field to another field. We can also take a value, just a string constant or a string that is made up of a mixture of constant and flexible string expansion variables using the \${ } syntax, that will be put in the field. We can also specify a default value in the case that the value evaluates to an empty string, or the from field is null or empty. Then the default-value will be used. Again we can use the flexible string expander here, the \${ } syntax and such. It can also do a type conversion. So going from whatever type the source data is in, which would be a string value or whatever the variable type is for a from field, it can convert that to any of these types before setting it in the target field.

Attribute Details

Field="..."	Required
Name of the field to copy value to.	
From-field="..."	Optional
Name of the field to copy value from.	
Value="..."	Optional
Simple value to copy in field.	
default-value="..."	Optional
Default value to copy in field, if the value evaluates to an empty string or the from field is null or empty.	
Type="..."	Optional
Type to convert to, can be: PlainString, String, BigDecimal, Double, Float, List, Long, Integer, Date, Time, Timestamp, Boolean, Object.	
set-if-null="true false"	Optional – defaults to true
Specifies whether or not to set fields that are null or empty.	
set-if-empty="true false"	Optional – defaults to true
If the source value, either from a value or from a field is empty, an empty-string, an empty list or a null value. In this case it's set to true. If you don't want to set it, if you want it to leave the target field alone when the source is empty, then you need to set this to false.	

Usage

```
<set from-field="productstoreId" field="newEntity.productstoreId"/>
```

<clear-field>

The <clear-field> tag clears/removes the specified field.

Attribute Details

map-name="..."	Optional
----------------	----------

The name of the map in the method environment. If not specified the field-name will be used to get the field from the method environment.

Field-name="..."	Required
------------------	----------

The name (key) of the map field to use.

Usage

```
<clear-field field-name="variantProduct"/>
```

<first-from-list>

The <first-from-list> tag will get the first entry from the given list and put it in the environment field with the given entry-name.

Attribute Details

Entry-name="..."	Required
------------------	----------

The name of the method environment field that will contain the first entry in the list.

List-name="..."	Required
-----------------	----------

The name of the method environment field that contains the list to get the first entry from.

Usage

```
<first-from-list entry-name="statusItem" list-name="contentStatus"/>
```

Miscellaneous Entity Operations

The following miscellaneous operations deal with obtaining primary or sequenced keys.

<sequenced-id-to-env>

The `<sequenced-id-to-env>` tag gets a sequenced ID from the Entity Engine (through the delegator) and puts it in the specified method environment field. The object will be a `java.lang.Long`, but can of course be converted to a `String`.

Attribute Details

`Sequence-name="..." Required`

The name of the sequence to pass to the delegator. The same name must always be used for sequenced IDs that will be used for a certain entity field otherwise non-unique keys may result.

`env-name="..." Required`

The name of the method environment field the sequenced ID will be put in.

`stagger-max="..." Optional`

By default this is one. But if you want to have sequenced IDs that are staggered, instead of consecutive, then you can set this to something like twenty. And then it will do a random staggering for each sequenced id. Instead of picking the next value all the time it will pick something between the next value and twenty away from the next value, if `stagger-max` is set to twenty. So that can be used to make the sequenced Ids more difficult to guess.

Usage

```
<sequenced-id-to-env sequence-name="OrderHeader" env-name="orderId"/>
```

<make-next-seq-id>

Whereas `<sequenced-id-to-env>` is the primary key sequencer, `<make-next-seq-id>` is the secondary key sequencer. So this would be something like an `orderId` for example, where we're sequencing an `orderId` automatically. And this would be something like an `orderItemSequenceId`, where we have a sub-sequence that varies for order item records related back to an `orderHeader`. So all of them will have the same `orderId`.

Attribute Details

`value-name="..." Required`

Specify the name of the entity for a sequenced-id-to-env preparing the primarySequencedId. The name of the entity is typically what we use for the sequenced name, but you can use anything you want, if you want to have different sets of sequences. The risk of course (in having many different sets of sequences for the same entity) is that unless you somehow prefix or suffix the value, you could have a key conflict. So we just use the entity name for these primary sequences.

`seq-field-name="..." Required`

The field that will have the sub-sequenced value. We use the seqId suffix on the field names in the OFBiz data model to denote that that field is a secondary sequenced ID and should therefore be maintained for this sort of operation.

`numeric-padding="..." Optional—defaults to 5`

Since these are eventually strings we do numeric-padding so that the sort is consistent. By default we pad it with five positions. For example 00001.

`increment-by="..." Optional—defaults to 1`

If you want to leave some space in the sub-sequence you can use a greater increment.

Usage

```
<make-next-seq-id value-name="newOrderItem"
                    seq-field-name="orderItemSeqId" />
```

Entity Find Operations

Entity Find Operations deal with retrieving data from the database.

<entity-one>

Performs a find by primary key lookup, returning a single GenericValue object if it's found, otherwise it returns null. This operation replaces the deprecated <find-by-primary-key> operation.

Attribute Details

`entity-name="..." Required`

Name of the entity to look for.

`value-name="..." Required`

Name of the variable to put the GenericValue in.

`use-cache="true|false"` Optional – defaults to `false`

Specifies whether or not the delegator's cache should be searched before going to the database. This results in much faster retrieval times, but can return stale data that is not the most current in the database.

`auto-field-map="true|false"` Optional – defaults to `true`

Specifies whether to look for all primary key field names in the current context as well as in the parameters map or not.

Usage

```
<entity-one entity-name="OrderHeader" value-name="orderHeader"
            auto-field-map="true"/>
```

<entity-and>

Performs a find-by-and lookup, returning a list of GenericValues if any are found, otherwise returning null. Uses name/value pairs, that will be used for the query and will all be "anded" together. The result-set-type by default is scroll which is flexible so we can go forward. This operation replaces the deprecated `<find-by-and>` operation and uses the `<field-map>` sub-element to pass in lookup criteria.

Attribute Details

`entity-name="..."` Required

Name of the entity to look for.

`list-name="..."` Required

Name of the variable to put the GenericValue in.

`use-cache="true|false"` Optional – defaults to `false`

Specifies whether or not the delegator's cache should be searched before going to the database. This results in much faster retrieval times, but can return stale data that is not the most current in the database.

`filter-by-date="true|false"` Optional – defaults to `false`

Look for from date and through date fields in the list of results coming back, and filters by the current date and time if set to true.

Usage

```
<entity-and entity-name="ProductFacility" list-name="products">
  <field-map env-name="parameters.facilityId" field-name="facilityId"/>
</entity-and>
```

<entity-condition>

Like <entity-and> returns a list of GenericValues if any are found, otherwise returns null.

Uses any of <condition-expression>, <condition-list> and <condition-object> sub-elements to specify the lookup criteria.

Attribute Details

entity-name="..."	Required
Name of the entity to look for.	
list-name="..."	Required
Name of the variable to put the GenericValue in.	
use-cache="true false"	Optional - defaults to false
Specifies whether or not the delegator's cache should be searched before going to the database. This results in much faster retrieval times, but can return stale data that is not the most current in the database.	
filter-by-date="true false"	Optional - defaults to false
Look for from date and through date fields in the list of results coming back and filters by the current date and time if set to true.	

Sub-Elements

The following sub-elements are used by the entity-condition operation to specify WHERE conditions, HAVING conditions, fields to select and fields to order by.

<condition-list>

A <condition-list> is a list of conditions that are combined with either and or or. The default is and. We can have condition-lists under condition-list, for building fairly complex condition trees, and we may also drop in condition-objects at any point.

Usage

```
<condition-list combine="and">
    <condition-expr field-name="partyId" operator="equals"
        env-name="parameters.partyId"/>
    <condition-expr field-name="roleTypeId" operator="equals"
        env-name="parameters.roleTypeId"/>
</condition-list>
```

The `<condition-expr>` tag basically combines a field, a SQL operator and a variable or a value. The variable is actually a field in the context : `env-name`. There are some options when to ignore the condition, namely if the `env-name` or value is empty or null.

<having-condition-list>

Similar to condition-list but relates to SQL grouping : having-condition runs after the grouping.

<select-fields>

Name of the field to select.

Usage

```
<select-field field-name="partyId"/>
```

<order-by>

Defines fields to order list by.

Usage

```
<order-by field-name="-changeDate"/>
```

<entity-condition> complete example:

```
<entity-condition entity-name="OrderHeaderAndRoleSummary"
    list-name="orderInfoList">
    <condition-list combine="and">
        <condition-expr field-name="partyId" operator="equals"
            env-name="parameters.partyId"/>
        <condition-expr field-name="roleTypeId" operator="equals"
            env-name="parameters.roleTypeId"/>
        <condition-expr field-name="orderTypeId" operator="equals"
            env-name="parameters.orderTypeId"/>
        <condition-expr field-name="statusId" operator="equals"
```

```
        env-name="parameters.statusId"/>
    <condition-expr field-name="orderDate" operator="greater-equals"
        env-name="parameters.fromDate"
        ignore-if-null="true"/>
    <condition-expr field-name="orderDate" operator="less-equals"
        env-name="parameters.thruDate"
        ignore-if-null="true"/>
</condition-list>
<select-field field-name="partyId"/>
<select-field field-name="roleTypeId"/>
<select-field field-name="totalGrandAmount"/>
<select-field field-name="totalSubRemainingAmount"/>
<select-field field-name="totalOrders"/>
<order-by field-name="partyId"/>
</entity-condition>
```

<entity-count>

The `<entity-count>` tag is very similar to the `<entity-condition>`. Specify the `entity-name`, optionally the `delegator-name` if you want to override that, and then the name of the variable to put the count in. You can do the same `condition-expr` (condition expression) or `condition-list` which can have `condition-expr` and other `condition-lists` underneath it for a tree of conditions that can be arbitrarily complex. You can also use the `having-condition-list`, this is the same as on the `entity-condition`. What this will do basically is, rather than doing a query and getting the results back, it will just count the results and put that number in the context in the variable named by the `count-name`.

Usage

```
<entity-count entity-name="ReturnAdjustment"
    count-name="returnCount">
    <condition-expr field-name="orderAdjustmentId" operator="equals"
        env-name="orderAdjustment.orderAdjustmentId"/>
</entity-count>
```

<get-related-one>

If we have a `GenericValue` object sitting in the context and we want to get a related entity following one of the type one relationships in the data model, then we can specify the name of the value here, `relation-name`, which is the name of the relationship between the two entities, and the `to-value` where it should put the result.

Attribute Details

value-name="..." Required
Name of a generic value sitting in the context from where you want to get a related-one generic value.
relation-name="..." Required
Name of the one-relation to use to relate a generic value in value-name, to a generic value in to-value.
to-value-name="..." Required
Name of a GenericValue where to put the result.
use-cache="true false" Optional – defaults to false
Specifies whether or not the delegator's cache should be searched before going to the database. This results in much faster retrieval times, but can return stale data that is not the most current in the database.

Usage

```
<get-related-one value-name="orderHeader"
                  relation-name="ProductStore"
                  to-value-name="productStore" use-cache="true"/>
```

<get-related>

<get-related> is just like <get-related-one>, except that instead of getting a single value back to put in the to-value, it gets a full List back. We start with the value object, and specify the name of the relationship. We can specify the name of a map that will restrain the query further, beyond the field mappings and their relationship.

We can also specify how you want to order it.

Attribute Details

value-name="..." Required
Name of a GenericValue sitting in the context from where we want to get related GenericValues.
relation-name="..." Required
Name of the one-relation to use to relate GenericValue in value-name to GenericValue in to-value.
list-name="..." Required
Name of a list to put the result in.

<code>map-name="..."</code>	Optional
Name of a map that will restrain the query further, beyond the field mappings and their relationship.	
<code>order-by-list-name="..."</code>	Optional
This will be a list sitting in the context that has string entries in it for each field that we want it to order by. Each field in the list, or each entry in the list, will just be a string with a field name. It can be preceded by a plus or a minus to specify an ascending or descending sort. The default is ascending sort, so we just put a minus in front of the field-name if we want it to be descending.	
<code>use-cache="true false"</code>	Optional – defaults to false
Specifies whether or not the delegator's cache should be searched before going to the database. This results in much faster retrieval times, but can return stale data that is not the most current in the database.	

Usage

```
<get-related value-name="order" relation-name="OrderItem"
             list-name="orderItems"/>
```

<order-value-list>

The `<order-value-list>` operation does not actually do anything with the database. It takes an existing List that has been returned and sorts it.

Attribute Details

<code>list-name="..."</code>	Required
Name of the list of generic value objects that we want to sort.	
<code>order-by-list-name="..."</code>	Required
This will be a list sitting in the context that has string entries in it for each field that you want it to order by. Each field in the list, or each entry in the list, will just be a string with a field name. It can be preceded by a plus or a minus to specify an ascending or descending sort for that. The default is ascending sort, so you just put a minus in front of the field-name if you want it to be descending.	
<code>to-list-name="..."</code>	Optional
Name of the output list. If it is empty, as it is optional, it will simply use the list-name. In other words it will take the ordered list and put it over top of the resource list.	

Usage

```
<order-value-list list-name="facilityLocationList"
                  order-by-list-name="facilityLocsOrdLst"/>
```

<filter-list-by-and>

The <filter-list-by-and> tag filters the given list by the fields in the specified map.

Attribute Details

list-name="..."	Required
The name of the method environment field that contains the list of GenericValue objects.	
to-list-name="..."	Optional
The name of the method environment field the filtered list will be put into. Defaults to the value of the list-name attribute (For example, it goes to the same place it came from, replacing the old List).	
map-name="..."	Optional
The name of a Map in the method environment that will be used for the entity fields. If no Map is used this will just make a copy of the List.	

Usage

```
<filter-list-by-and map-name="itemFilterMap"
                     list-name="orderReadyToPickInfo.orderItemShipGrpInvResList"
                     to-list-name="perItemResList"/>
```

<filter-list-by-date>

The <filter-list-by-date> tag filters the given list by the valid date using the from and thru dates in each value object.

Attribute Details

list-name="..."	Required
The name of the method environment field that contains the list of GenericValue objects.	
to-list-name="..."	Optional

The name of the method environment field the filtered list will be put into.
Defaults to the value of the list-name attribute (For example, it goes to the same place it came from, replacing the old list).

`valid-date-name="..." Optional`

The name of a field in the method environment date to filter by. Defaults to now.

`from-field-name="..." Optional – defaults to "fromDate".`

The name of the GenericValue field to use as the beginning effective date.

`thru-field-name="..." Optional – defaults to "thruDate".`

The name of the GenericValue field to use as the beginning effective date.

`all-same="true|false" Optional – defaults to true`

Specifies whether or not all GenericValue objects in the list are of the same entity.

Usage

```
<filter-list-by-date list-name="emailAddressesAll"
                     from-field-name="fromDate" to-list-name="emailAddresses"/>
```

Entity Value Operations

These operations deal with persisting, updating and removing values to and from the database.

<make-value>

The make-value tag uses the delegator to construct an entity value. The resulting value will not necessarily exist in the database, but will simply be assembled using the entity-name and fields map. The resulting GenericValue object will be placed in the method environment using the specified value-name.

Attribute Details

`value-name="..." Required`

The name of the method environment field that contains the GenericValue object.

`entity-name="..." Required`

The name of the entity to construct an instance of.

`map-name="..." Required`

The name of a map in the method environment that will be used for the entity fields.

Usage

```
<make-value value-name="newEntity" entity-name="OrderHeader"/>
```

<clone-value>

The <clone-value> tag makes a copy of the value in the method environment field specified by value-name. The resulting GenericValue object will be placed in the method environment using the specified new-value-name.

Attribute Details

value-name="..."	Required
------------------	----------

The name of the method environment field that contains the GenericValue object.

new-value-name="..."	Required
----------------------	----------

The name of the method environment field that will contain the new GenericValue object.

Usage

```
<clone-value value-name="partyContactMech"
             new-value-name="newPartyContactMech"/>
```

<create-value>

The <create-value> tag persists the specified GenericValue object by creating a new instance of the entity in the datasource.

An error will result if an instance of the entity exists in the datasource with the same primary key.

Attribute Details

value-name="..."	Required
------------------	----------

The name of the method environment field that contains the GenericValue object.

do-cache-	Optional - defaults to true
-----------	-----------------------------

clear="true false"	
--------------------	--

Clears the cache.

Usage

```
<create-value value-name="newEntity"/>
```

<store-value>

The `<store-value>` tag persists the specified `GenericValue` object by updating the instance of the entity in the datasource. An error will result if an instance of the entity does not exist in the datasource with the same primary key.

Attribute Details

<code>value-name="..."</code>	Required
The name of the method environment field that contains the <code>GenericValue</code> object.	
<code>do-cache-</code> <code>clear="true false"</code>	Optional – defaults to <code>true</code>
Clears the cache.	

Usage

```
<store-value value-name="orderHeader" />
```

<refresh-value>

Forces a value to refresh from the database.

Attribute Details

<code>value-name="..."</code>	Required
The name of the method environment field that contains the <code>GenericValue</code> object.	
<code>do-cache-</code> <code>clear="true false"</code>	Optional – defaults to <code>true</code>
Clears the cache.	

Usage

```
<refresh-value value-name="newEntity" />
```

<remove-value>

The `remove-value` tag removes the specified `GenericValue` object by removing the instance of the entity in the datasource.

Attribute Details

<code>value-name="..."</code>	Required
The name of the method environment field that contains the GenericValue object.	
<code>do-cache-</code> <code>clear="true false"</code>	Optional – defaults to <code>true</code>
Clears the cache.	

Usage

```
<remove-value value-name="someEntity"/>
```

<remove-related>

Given a value name and a relationship name, this operation follows the relationship and removes all related records, whether they be a type one or type many relationship.

For a type one relationship it will remove a single record if it exists, and for a type many relationship it will remove all the records that are related to it.

This of course can be dangerous. For example if we have a product-type entity and we do a remove-related with a certain product type value object here, and the relation name product here, it will remove all products of that type. This is of more value when we are doing something like removing an order or removing a product, and we want to remove all related elements before removing the product to resolve foreign key issues. Usually the best practice for that, is to do a remove-related on certain types of information related to the product, and then try to remove the product but not remove all related tables.

In many cases, if the product has been ordered for example, then you do not want to remove the product, and so you can do all these remove-related(s).

Attribute Details

<code>value-name="..."</code>	Required
The name of the method environment field that contains the GenericValue object.	
<code>relation-name="..."</code>	Required
Name of a relation to use to remove related records.	
<code>do-cache-</code> <code>clear="true false"</code>	Optional – defaults to <code>true</code>
Clears the cache.	

Usage

```
<remove-related value-name="content" relation-name="ContentRole"/>
```

<remove-by-and>

The <remove-by-and> tag uses the delegator to remove entity values from the datasource and is constrained by "anding" the fields passed in the Map. Make sure the Map contains something, or all values will be removed.

Attribute Details

entity-name="..."	Required
The name of the entity to remove instances of.	
map-name="..."	Required
The name of a map in the method environment that will be used for the entity fields.	
do-cache-clear="true false"	Optional – defaults to true
Clears the cache.	

Usage

```
<remove-by-and entity-name="ProductPrice"
                map-name="productFindContext"/>
```

<set-pk-fields> / <set-nonpk-fields>

Looks for each primary key / non primary key field in the named map and if it exists there, it will copy it into the named value object.

Attribute Details

value-name="..."	Required
The name of the method environment field that contains the GenericValue object.	
map-name="..."	Required
The name of a map in the method environment that will be used for the entity fields.	
set-if-null="true false"	Optional – defaults to true
Specifies whether or not to set fields that are null or empty.	

Usage

```
<set-nonpk-fields map-name="parameters" value-name="newEntity"/>
<set-pk-fields map-name="parameters" value-name="newEntity"/>
```

<store-list>

The <store-list> tag uses the delegator to store all entity values in the list. This is different than storing a single value using the <store-value> operation in that values in the List will be inserted if they do not exist, or updated if they do.

Attribute Details

list-name="..."	Required
The name of the method environment field that contains the list of GenericValue objects.	
do-cache-clear="true false"	Optional – defaults to true
Clears the cache.	

Usage

```
<store-list list-name="returnItems"/>
```

Control Operations

Control operations allow us to iterate through Lists and Maps, add and check errors, and return from the Simple Method.

<iterate>

The operations contained by the iterate tag will be executed for each of the entries in the List, and will make the current entry available in the method environment by the entry-name specified. This tag can contain any of the simple-method operations, including the conditional/if operations. Any simple-method operation can be nested under the iterate tag.

Attribute Details

entry-name="..."	Required
The name of the method environment field that will contain each entry as we iterate through the list.	
list-name="..."	Required
The name of the method environment field that contains the list to iterate over.	

Usage

```
<iterate list-name="orderItems" entry-name="orderItem">
  ...
</iterate>
```

<iterate-map>

The operations contained by the iterate-map tag will be executed for each of the entries in the map. It will run all of the operations underneath the iterate-map-element for each of the entries in the given map, setting the key for that entry and the key-name variable, and the value for that entry and the value-name variable. This tag can contain any of the simple-method operations, including the conditional/if operations. Any simple-method operation can be nested under the iterate-map tag.

Attribute Details

Key-name="..."	Required
Name of the variable to put the key in.	
Value-name="..."	Required
Name of the variable to put the value in.	
Map-name="..."	Required
Name of the map to use.	

Usage

```
<iterate-map map-name="inventoryItemQuantities"
  key-name="inventoryItemId" value-name="quantityNeeded">
  ...
</iterate-map>
```

<check-errors>

The message lists from invoking are not checked until the <check-errors> tag is used.

The named List is checked and if it contains any messages they are put in the servlet request object, and the specified error code is returned to the control servlet.

Attribute Details

Error-code="..."	Optional – defaults to error
Defaults to error.	
Error-list-name="..."	Optional – defaults to error_list
The name of the list in the method environment to check for error messages.	

Usage

```
<check-errors/>
```

<add-error>

Adds an error message to the given error list from either an inline message or a message from a properties file using the sub elements <fail-message> or <fail-property>.

Attribute Details

Error-list-name="..."	Optional – defaults to error_list
The name of the list in the method environment to add the error message to.	

Usage

```
<add-error>
  <fail-property resource="PartyUiLabels"
    property="PartyPermissionErrorForThisParty"/>
</add-error>
```

<return>

Returns immediately from the simple-method with the given response code string.

Attribute Details

response-code="..."	Optional – defaults to success
The string to return as a response code.	

Usage

```
<return/>
```

If Conditions

These operations allow us to control the flow through the Simple Method depending on simple if tests.

<if-validate-method>

The operations contained by the <if-validate-method> tag will only be executed if the validate method returns true. <if-validate-method> calls a static Java method that takes a `String` and returns a `boolean`. This tag can contain any of the simple-method operations, including the conditional/if operations.

Attribute Details

<code>Field-name="..."</code>	Required
The name of the map field that will be validated.	
<code>method="..."</code>	Required
The name of the method that will be called to validate the field. It must be a static method that takes a single <code>String</code> parameter and returns a <code>boolean</code> .	
<code>Map-name="..."</code>	Optional
The name of the method environment field that contains the map that the field to be validated will come from. If not specified the <code>field-name</code> will be treated as a method environment field name (an env-name).	

Usage

```
<if-validate-method field-name="${answerFieldName}"  
                   method="isCreditCard">  
    ... success operations ....  
    <else>  
        ... failure operations ...  
    </else>  
</if-validate-method>
```

<if-instance-of>

Checks if the field is an instance of the name class.

Attribute Details

field-name="..."	Required
------------------	----------

The name of the map field that will be validated as being an instance of the named class.

class="..."	Required
-------------	----------

The name of the class that the named instance in field-name is supposed to belong.

Usage

```
<if-instance-of field-name="parameters.categories"
    class="java.util.List">
    ... success operations ...
    <else>
        ... failure operations ...
    </else>
</if-instance-of>
```

<if-compare>

The operations contained by the `<if-compare>` tag will only be executed if the comparison returns `true`. This tag can contain any of the simple-method operations, including the conditional/if operations.

Attribute Details

field-name="..."	Required
------------------	----------

The name of the map field that will be compared.

value="..."	Required
-------------	----------

The value that the field will be compared to. Must be a String, but can be converted to other types.

map-name="..."	Optional
----------------	----------

The name of the method environment field that contains the map that the field to be validated will come from. If not specified the field-name will be treated as a method environment field name (an env-name).

Usage

```
<if-compare field-name="product.productTypeId" operator="not-equals"
    value="DIGITAL_GOOD"/>
    ... success operations ...
    <else>
        ... failure operations ...
    </else>
</if-compare>
```

<if-compare-field>

The operations contained by the <if-compare-field> tag will only be executed if the comparison returns true. This tag can contain any of the simple-method operations, including the conditional/if operations.

Attribute Details

field-name="..."	Required
The name of the map field that will be compared.	
map-name="..."	Optional
The name of the method environment field that contains the map that the field to be validated will come from. If not specified the field-name will be treated as a method environment field name (an env-name).	
to-field-name="..."	Optional
The name of the to-map field that the main field will be compared to. If left empty, will default to the field-name.	
to-map-name="..."	Optional
The name of the method environment field that contains the map that the field to be compared will come from. If left empty will default to the method environment. It does not default to the map-name because that would make it impossible to compare a map field to an environment field.	

Usage

```
<if-compare-field field-name="inventoryFacilityId" map-name="store"
                   operator="not-equals"
                   to-field-name="oldFacilityId">
    ... success operations ....
    <else>
        ... failure operations ...
    </else>
</if-compare-field>
```

<if-empty> / <if-not-empty>

The operations contained by the <if-empty> / <if-not-empty> tag will only be executed if the map field is empty / not empty. These tags can contain any of the simple-method operations, including the conditional/if operations.

Attribute Details

field-name="..."	Required
	The name of the map field that will be compared.
map-name="..."	Optional
	The name of the method environment field that contains the map that the field to be validated will come from. If not specified the field-name will be treated as a method environment field name (an env-name).

Usage

```
<if-empty field-name="parameters.roleTypeId">
    ... success operations ...
    <else>
        ... failure operations ...
    </else>
</if-empty>
```

Conditions:

Nested conditions are acceptable in Minilang. Consider the following example:

```
<if>
    <condition><not><if-empty
        field-name="parameters.groupName"/>
    </not></condition>
    <then>
        <entity-one entity-name="PartyGroup"
            value-name="partyGroup"/>
        <if>
            <condition><if-compare-field
                field-name="partyGroup.groupName"
                operator="not-equals"
                to-field-name="parameters.groupName"/>
            </condition>
            <then>
                <set-nonpk-fields
                    value-name="partyNameHistory"
                    map-name="partyGroup"/>
                <create-value value-name="partyNameHistory"/>
            </then>
        </if>
    </then>
</if>
```

Other Operations

These operations allow us to add Debug messages, calculate values or obtain the Timestamp value for this instant.

<log>

The <log> tag logs a message used by the OFBiz Debug class, which uses Log4J to log to the console, a file, or some other location. The message is a concatenation of the message attribute and then all of the field and string sub-element values in the order they are specified. Possible levels, which relate to the logging levels set in debug.properties are: verbose, timing, info, important, warning, error, fatal, always.

Usage

```
<log level="info" message="Contact mech created with id ${newValue.  
contactMechId}"/>
```

<now-timestamp-to-env>

The <now-timestamp-to-env> tag creates a java.sql.Timestamp object with the current date and time in it and puts it in a field in the method environment.

Usage

```
<now-timestamp-to-env env-name="nowTimestamp" />
```

<now-date-to-env>

The <now-date-to-env> tag creates a java.sql.Date object with the current date in it and puts it in a field in the method environment.

Usage

```
<now-date-to-env env-name="nowDate" />
```

<set-current-user-login>

The <set-current-user-login> tag sets the UserLogin GenericValue object to be used for authentication for the rest of the method. This is mostly used for calling services, and so on.

Usage

```
<set-current-user-login value-name="createdUserLogin" />
```

<calculate>

The <calculate> tag performs the specified calculation and puts the result in an object in the field of the specified Map.

The type of the object can be specified with the type attribute, this defaults to Double. The calculate tag can contain <calcop> and <number> tags, and the <calcop> tag can also contain these two tags to enable nested calculations. The operator specifies the operation to perform on the given field and nested calcops and numbers. It must be one of the following: get | add | subtract | multiply | divide | negative.

Attribute Details

field-name="..."	Required
The name (key) of the Map (or env if map-name is empty) field to use.	
map-name="..."	Optional
The name of the map in the method environment. If not specified the field-name will be used to get the field from the method environment.	
type="..."	Optional – defaults to Double
Specifies the type of the object. Can be String Double Float Long Integer.	
rounding-mode="..."	Optional – defaults to Double
Rounding mode for BigDecimal calculation, primarily for divide operation.	
rounding-mode="..."	Optional – defaults to Double
Rounding mode for BigDecimal calculation, primarily for divide operation. Can be Ceiling Floor Up Down HalfUp HalfDown HalfEven Unnecessary.	
decimal-scale="..."	Optional – defaults to 2
Initial scale to use for the internal BigDecimal. Defaults to 2 for monetary calculations.	

Usage

```
<set field="productCost" value="0" type="Double" />
<calculate field-name="productCost" type="Double" decimal-scale="6">
    <calcop field-name="laborCost.cost" operator="add">
        <calcop field-name="materialsCost.cost" operator="get"/>
        <calcop field-name="routeCost.cost" operator="get"/>
        <calcop field-name="otherCost.cost" operator="get"/>
    </calcop>
</calculate>
```

Index

Symbols

<complex-alias> element
nested <complex-alias> element 212
operators 212, 213

<entity> element, entities
<field> element 188
<field> element, database independence 188
<field> element, type attribute 189
<index> element 194, 195
<index> element,
 foreign keys index 195, 196
<prim-key> element 189
<relation> element 189
<relation> element, database updating 193
<relation> element, field creating 190, 191
<relation> element, relation
 accessing 191-193
<relation> element, relation creating 190
about 187, 188

<section> element, screen widget
<actions> element 76, 77
<fail-widgets> element 78
<if-condition> element 76
<widgets> element 77, 78
conditional screen widget, viewing 79
context, utility objects 80
controller.xml, element order 75
control servlet, informing about screen
 widget 75
if-then-else structure 75
minimum <section> 78
nesting 85-88
parameters, sending with requests 79
screen widget, context 79

screen widget, variables 79

<view-entity> element
<alias-all> element 205, 206
<alias> element 202-204
<member-entity> element 201, 202
<relation> element 209
<view-link> element 206, 207

A

alternative targets, form widget
viewing 126

Apache HTTP server
about 392
configuring, to use mod_proxy_ajp 393, 394
downloading 392
installing 392
secure pages, getting 395
self-signed SSL certificate, creating 396
SSL, configuring 395

asynchronous services
about 304
Job scheduler, using 305, 306

B

beanshell, converting to Java event
about 57-59
clean extension strategy, employing 59, 60
testing 60

C

call operations
<call-map-processor> 402
<call-map-processor>, attributes 403
<call-map-processor>, usage 403

<call-service-asynch> 403
<call-service-asynch>, attributes 403
<call-service-asynch>, usage 404
<call-service> 403
<call-service>, attributes 403
<call-service>, usage 404
<set-service-fields> 404
<set-service-fields>, attributes 404
<set-service-fields>, usage 404
about 402
complex permissions, service engine
and simple permissions 335
entity PlanetReview, creating 324
form widgets, creating 322, 324
permission services, defining 327, 328
permission services,
 implementing 328-330
request, creating 322
screen, creating 322, 324
services, defining 325
services, implementing 325-327
view maps, creating 322
working with 331-334
components, OFBiz
core application components 53
creating 55
files, moving from party component 56, 57
framework components 53
referencing 54
using 55
conditions 433
context
utility objects 80, 81
controller, in OFBiz 65, 66
control operations
 <add-error> 429
 <add-error>, attributes 429
 <add-error>, usage 429
 <check-errors> 428
 <check-errors>, attributes 429
 <check-errors>, usage 429
 <iterate-map> 428
 <iterate-map>, attributes 428
 <iterate-map>, usage 428
 <iterate> 427
 <iterate>, attributes 427
 <iterate>, usage 428
<return> 429
<return>, attributes 429
<return>, usage 429
about 427
control servlet
defining, in Webapp 152
programming 156
servlet, about 156
using 153
utility object, GenericDelegator
 object 154, 155
utility object, GenericDispatcher object 155
utility objects, defining 154

D

data accessing, BeanShell script used
example, playground setting up 218
example, playground testing 222, 223
generic screen, form widget creating 220
generic screen, FreeMarker
 file creating 220, 221
generic screen, publishing 222
generic screen, screen widget
 creating 219, 220
script processor 218, 219
database, Java events
working with 284
database record deleting,
 GenericValue object 226
database record retrieving,
 GenericValue object
about 226
AND joiner 232
comparison operators
 (*EntityComparisonOperator*) 228, 229
condition, styles 238
condition joiners (*EntityJoinOperator*) 232
condition lists 231
conditions, in OFBiz 228
conditions, joining by AND 238
conditions, joining by OR 239, 240
date condition 241
fields, selecting 234
filterByOr method 251
findByCondition method, parameters 227
fromDate, field name specifying 250

nested condition lists 233
one-to-many relations 244, 245
one-to-one relations 243, 244
OR joiner 233
post-query, filterByAnd method 250
post-query, filtering by conditions 247, 248
post-query, filtering by date 248, 249
post-query, operations 245
post-query, ordering 246, 247
post-query, other filtering methods 250
records, finding by conditions 227
records retrieved, counting 241
related records, getting 243
retrieved records, counting 240
retrieved records, ordering 234, 239
set-value operators 230
single-value operators 229
text-value operators 231
thruDate, field name specifying 250
database record updating, GenericValue object 225
debugging, techniques
information, writing to logs 375, 376
logging 374
logging, console.log 374
logging, ofbiz.log 374
logging Minilang 376, 377
logs, configuring 374, 375
logs viewing, Webtools used 375
debugging Javacode
about 378
remote debugger, configuring 379-381
VM parameters, passing 378
derby data files
archiving 51
backing up 51
database, updating 49
deleting 49
restoring 49, 52
web server work files, backing up 51
web server work files, restoring 52
development, tips 397, 398
dynamic view entities
about 252
and functions 263
EntityFindOptions object 257
example 252-255
left outer joins 256
lookup, performing 257

E

ECA
about 308
Entity Event Condition Actions (EECAs) 310, 311
rules 308
Service Event Condition Actions (SECAs) 308-310
Eclipse
downloading 16
Eclipse project, setting up 16-19
entities
entity relation, one-to-many relations 196
entity relation, one-to-one relations 196
entity relation, types 196
one-to-many relations, one-to-one relations
 inverse 197-199
one-to-one relations, with no foreign keys
 199, 200
entities, extending
<extend-entity> element used 213, 214
entities, OFBiz
<entity> element 187, 188
datasources 177, 178
drop-down, creating 184-186
drop-down, populating 186
entity, assigning to entity group 180
entity, defining 179
entity, loading into entity engine 180, 181
entity, using 183
entity, viewing 181, 182
entity delegator 178
entity group 179
fields, referencing 176
Relational Database Management Systems (RDBMS) 176
relational model 177
value, expiring 187
value, un-expiring 187

entity engine cache, OFBiz
 using 251, 252
entity find operations
 <condition-list> 416
 <condition-list>, usage 417
 <entity-and> 415
 <entity-and>, attributes 415
 <entity-and>, usage 416
 <entity-condition> 416
 <entity-condition>, attributes 416
 <entity-condition>, example 417, 418
 <entity-count> 418
 <entity-count>, usage 418
 <entity-one> 414
 <entity-one>, attributes 415
 <entity-one>, usage 415
 <filter-list-by-and> 421
 <filter-list-by-and>, attributes 421
 <filter-list-by-and>, usage 421
 <filter-list-by-date> 421
 <filter-list-by-date>, attributes 422
 <filter-list-by-date>, usage 422
 <get-related-one> 418
 <get-related-one>, attributes 419
 <get-related-one>, usage 419
 <get-related> 419
 <get-related>, attributes 419, 420
 <get-related>, usage 420
 <having-condition-list> 417
 <order-by> 417
 <order-by>, usage 417
 <order-value-list> 420
 <order-value-list>, attributes 420
 <order-value-list>, usage 421
 <select-fields> 417
 <select-fields>, usage 417
 sub-elements 416
EntityListIterator
 paginating through 258
entity operations
 <make-next-seq-id> 413
 <make-next-seq-id>, attributes 414
 <make-next-seq-id>, usage 414
 <sequenced-id-to-env> 413
 <sequenced-id-to-env>, attributes 413
 <sequenced-id-to-env>, usage 413

entity value operations
 <clone-value>, attributes 423
 <clone-value>, usage 423
 <create-value> 423
 <create-value>, attributes 423
 <create-value>, usage 423
 <make-value> 422
 <make-value>, attributes 422
 <make-value>, usage 423
 <refresh-value>, attributes 424
 <refresh-value>, usage 424
 <remove-by-and> 426
 <remove-by-and>, attributes 426
 <remove-by-and>, usage 426
 <remove-related> 425
 <remove-related>, attributes 425
 <remove-related>, usage 426
 <remove-value> 424
 <remove-value>, attributes 425
 <remove-value>, usage 425
 <set-nonpk-fields>, attributes 426
 <set-nonpk-fields>, usage 427
 <set-pk-fields>, attributes 426
 <set-pk-fields>, usage 427
 <store-list> 427
 <store-list>, attributes 427
 <store-list>, usage 427
 <store-value> 424
 <store-value>, attributes 424
 <store-value>, usage 424
 about 422
environment operations
 <clear-field> 412
 <clear-field>, attributes 412
 <clear-field>, usage 412
 <first-from-list> 412
 <first-from-list>, attributes 412
 <set> 411
 <set>, attributes 411
 <set>, usage 412
 usage 412
Event Condition Actions. See ECA
event operations
 <session-to-field> 410
 <session-to-field>, attributes 410
 <session-to-field>, usage 410

existing screen widgets, reusing 145

F

feedback, Java events

conventions, testing 275-277
message placeholders, conventions for 274
sending, to end-user 274

FOP 367

Formatting Objects Processor. *See FOP*

form processing, request events used 116, 117

form widget

about 111
containing screen widget, creating 112, 113
creating 112-114
CSS style sheet file maincss.css, requisites 115
files 112
form, processing 118, 119
form, submitting 118, 119
form attributes 114, 115
Java events 117, 118
JavaScript library selectall.js, requisites 116
list type form widget 119
locations 112
multi type form widget 122
referencing 113
row-level actions 128
two-target forms, alternative targets 126
using, minimal requirements 115, 116
viewing 114

form widget, creating

about 112-114
containing screen widget, creating 112, 113
form attributes 114, 115
form widget, requirements 115
referencing 113
viewing 114

FreeMarker

about 142, 360-362
as decorator templates 142, 143
dynamically created list variables, displaying 143, 144
iterating, through list 144

FreeMarker, using

about 360-362
Java, calling 364, 365
OFBiz transform tags 362
OFBiz transform tags,
 <@ofbizContentUrl> 363
OFBiz transform tags,
 <@ofbizCurrency> 364
OFBiz transform tags,
 <@ofbizUrl> 362, 363

functions and dynamic view entities 263

G

GenericValue object

about 223
database record, creating 224
database record, deleting 226
database record, retrieving 226
database record, updating 225
map, uses 224

H

header creating, screen widgets as templates

creating 98
using 99

I

if conditions

<if-compare-field> 432
<if-compare-field>, attributes 432
<if-compare-field>, usage 432
<if-compare> 431
<if-compare>, attributes 431
<if-compare>, usage 431
<if-empty> 432
<if-empty>, attributes 433
<if-empty>, usage 433
<if-instance-of> 430
<if-instance-of>, attributes 431
<if-instance-of>, usage 431
<if-not-empty> 432
<if-not-empty>, attributes 433
<if-not-empty>, usage 433
<if-validate-method> 430

<if-validate-method>, attributes 430
<if-validate-method>, usage 430
about 430

interfaces, implementing
about 303, 304
implemented attributes, overriding 304

J

Java code
debugging 378

Java code, creating for service 288, 289

Java Development Kit. *See* **JDK 5.0**

Java events
about 265, 266
database, working with 284
feedback, sending to end-user 274
form widget 117, 118
parameters, handling 277-279
security, dealing with 271-274
security and access control 266

Java Platform Debugger Architecture (JPDA) 378

JDK 5.0
downloading 14
installing 15

L

large screen widgets, organizing into smaller screens 88, 89

list type, form widget
containing screen widget, creating 119, 120
form, publishing 121
form processing code, adding 120, 121
list type form, viewing 122

localized messages, Java events
accessing 279-281
multiple languages, catering for 282-284
parameterizing messages 281, 282

M

memory allocation, OFBiz
logs, seeing in real time 25
OFBiz, starting 24

start-up, problems 25
start-up process, verifying 24, 25

menu widget
about 134
adding, in screen widgets 135
adding, via decorator
screen widget 135, 136
conditional-menu items 137-139
“ConditionalScreen” screen widget, adding 136, 137
creating 134, 135
menu item attributes 135
pre-processing actions 139-141
sub-menus 137, 138

Mini-Language. *See* **Minilang**

Minilang
<validate-method> 344
drawbacks 337
parameters, conversion 343, 344
security, checking 348
simple event example, validating 344-348
simple events 342, 343
simple map processor 343
simple service 339
syntax 338
using, in screen widgets 350, 351
validation 343, 344
XML coding, tools 339

Minilang, invoking from

Beanshell, calling 350
Java methods, calling 350
services, calling 349
simple methods, calling 349

mod_proxy_ajp 393

model, in OFBiz 64

Model-View-Controller (MVC) 63-66

multiple checks, combining 319

multi type, form widget

containing screen widget, creating 122, 123
data, loading for form 123, 124
form, publishing 124
form-processing logic, creating 125
multi type form, viewing 125

MVC, in plain english 63, 64

MVC framework 9

N

nested checks 320

O

OFBiz

accessing, patterns 26
components 53, 54
components, creating 55
components, referencing 54
configuring, to use AJP connector 394
controller 65
control servlet 151
database used 20
debugging, techniques 373
development, tips 397, 398
e-commerce application, accessing 27
ECA rules 308
entities 176
entity engine cache, using 251, 252
form widget 111
FreeMarker 142
Java events 265, 266
look and feel, modifying 353-355
Minilang 337
model 64
Model-View-Controller (MVC) 63, 66
request, firing 26, 27
screen widgets 69
seed data, loading 20
service engine 287
service engine, settings 305
structure 53
SVN workspace 13, 14
utilities 365
view 65
webapps 60

OFBiz, accessing patterns

about 26
webapp e-commerce 28-32
webapp order 32-34

OFBiz, compiling 20

OFBiz, look and feel

about 353, 355
back office screens,
application header 359, 360

back office screens, applications bar

(appbar) 358, 359

back office screens, central area 359

back office screens, changing 357, 358

back office screens, footer 360

back office screens, header 358

customer facing site, changing 355-357

OFBiz, running

memory, allocating 22-24

OFBiz, switching off 25, 26

OFBiz code, downloading 10

OFBiz ready to launch, downloading 16

OFBiz stock database, derby used

compilation process 20

data, loading 20

data loading, process 21

derby data files, backing up 22

installation process, verifying 20

OFBiz, compiling 20

OFBiz utilities

about 365

debug 366

UtilDateTime 366

UtilMisc 366

UtilValidate 366

other operations

<calculate> 435

<calculate>, attributes 435

<calculate>, usage 435

<log> 434

<log>, usage 434

<now-date-to-env> 434

<now-date-to-env>, usage 434

<now-timestamp-to-env> 434

<now-timestamp-to-env>, usage 434

<set-current-user-login> 434

<set-current-user-login>, usage 434

outputting

different formats 367

PDF 367-370

P

paginating

through EntityListIterator 258

through party records, example 258-262

parameters, Java events

handling 277-279

PDF
outputting 367-370

permissions, service engine
_ADMIN permissions 316, 317
complex permissions 321
multiple checks, combining 319
nested checks 320
role checks 317-319
simple permissions 313-315
two-part permissions 316

playground example, data accessing
generic screen 219, 220
script processor 218, 219
setting up 218
testing 222

programming, control servlet
auth attribute 162, 163
default responses 168
direct-request attribute 163, 164
event sub-element 165
handlers, specifying 159
https attribute 160, 161
Java events 165
learning application,
 logging in 156-158
non-responses 168
request-redirect-non param responses 171
request-redirect responses 170, 171
request instruction 159
request maps 160
request maps, https attribute 162
request maps, security attributes 160
request maps, uri attribute 160
request responses 169
response element, attributes 166, 168
view maps, defining 172
view responses 169

project, managing Subversion (SVN) used
about 382
branch, creating 387, 388
bug fixes, getting 388
code copy, importing 385, 386
code copy, preparing 384, 385
ofbiz\current, branching 387
OFBiz code, checking 384
OFBiz current, changes merging 391, 392

OFBiz current, updating 388-391
repository, preparing 383

R

request events
form, processing 116, 117

role checks 317-319

row-level actions, form widget
about 128
containing screen, creating 129, 130
form, creating 129
form, publishing 130
form, viewing 130

S

screen widgets
<section> element 74
anatomy 74
appheader, adding 148-150
as templates 97
creating 71
files and location 71
global context 90-92
integrating with FreeMarker 93, 94
large screen, organizing into
 smaller screens 88, 89
menu widget 134
outer context, making visible to
 nested ones 92, 93
screen widget view handler, using 71
user interface labels 146, 147
webapp equipping, screen widget view
 handler used 69-71

screen widgets, creating
control servlet, informing 72, 73
defining 71
flow pattern 73
referencing 73
viewing 74

**screen widgets, integrating with
FreeMarker 93, 94**

screen widgets as templates
approaches using, decorators
nesting 107-109
bottom-up approach (vertical stack),
 decorators nesting 106, 107

content slot, adding to decorator 103
 decorator, viewing 101, 102
 decorators, nesting 104
 decorator screen widgets, used for templating 100
 first half header, creating 102
 footer, creating 98
 footer, using 99
 header, creating 98
 header, using 99
 multi-slot decorator, using 103, 104
 multi-slot decorator, viewing 104
 multiple content slots 102
 second half header, creating 102
 templating 97
 top-down approach (delegation),
 decorators nesting 104-106
 XHTML decorator screen, creating 100
 XHTML decorator screen, using 100, 101
 XHTML document, viewing 99, 100
screen widget view handler, using 71
security, Java events
 dealing with 271-274
security, Minilang 348
security and access control, Java events
 security groups, assigning to user logins 271
 security groups, serving as access levels 269
 security permissions, containing within security groups 270, 271
 security permissions, serving as individual secured areas 270
 user logins 267, 268
service
 asynchronous services 304
 calling, from Java code 299-302
 defining 288
 DispatchContext 296
 interface, implementing 303, 304
 invoking, ways 302
 Java code, creating 288, 289
 Job scheduler, using 305, 306
 naming 307
 reference 307
 running 307
 security and access control 297-299
 synchronous services 304
 testing 289-291
service, calling from Java code 299-302
service, naming 307
service, running 307
service, security and access control 297-299
service calls
 <call-bsh> 407
 <call-bsh>, attributes 407
 <call-bsh>, usage 407
 <call-class-method> 409
 <call-class-method>, attributes 409
 <call-class-method>, usage 409
 <call-object-method> 408
 <call-object-method>, attributes 408
 <call-object-method>, usage 409
 <call-simple-method> 407
 <call-simple-method>, attributes 408
 <call-simple-method>, usage 408
 <result-to-field> 405
 <result-to-field>, attributes 405
 <result-to-field>, usage 405
 <result-to-request> 405
 <result-to-request>, attributes 406
 <result-to-request>, usage 406
 <result-to-result> 406
 <result-to-result>, attributes 406
 <result-to-result>, usage 407
 <results-to-map> 404
 <results-to-map>, attributes 405
 <results-to-map>, usage 405
service engine. *See service*
 permissions 313-315
service operations
 <field-to-result> 410
 <field-to-result>, attributes 410
 <field-to-result>, usage 410
service parameters
 about 291
 input parameters (IN) 291, 292
 optional and compulsory parameters 295
 output parameters (OUT) 293, 294
 special unchecked parameters 295
 two way parameters (INOUT) 294
service reference 307
simple events, Minilang 342, 343

simple method
 <simple-method>, attributes 401, 402
 <simple-method>, usage 402
 <simple-methods> 401
simple permissions
 about 313-315
 and complex permissions 335
simple service, Minilang
 about 339
 defining 340
 simple method, writing 340-342
structure, OFBiz
 about 53
 beanshell, converting to
 Java event 57-59
 component, using 55
 components 53, 54
 components, referencing 54
 OFBiz component, creating 55
 webapps 60
Subversion (SVN) 9
SVN
 using, to get OFBiz 11-13
synchronous services 304

T

testing, service 289-291
TortoiseSVN
 about 10
 downloading 10
two-part permissions 316, 317
two-target forms, form widget
 about 126
 containing screen, creating 127
 form, creating 126
 publishing 127
 viewing 128

U

utility objects, in context
 about 80
 availableLocales 81
 delegator 81
 dispatcher 81
 globalContext 80, 85

local (private) context 82, 83
locale 81
loosely typed variables, in BeanShell 83
map variables values, accessing 84, 85
nullField 80
parameters 81
screens 80
security 81
userLogin 81
variables values, accessing 84

V

validating, simple map processor used 343
view, in OFBiz 64
view entity
 <alias-all> element 205, 206
 <alias> element 202-204
 <complex-alias> element 212
 <member-entity> element 201, 202
 <relation> element 209
 <view-link> element 206, 207
 <view-link> element, inner joins versus
 outer joins 207, 208
about 200
arithmetic aggregate functions 210
complex aliases 211
complex aliases, nested <complex-alias>
 element 212
complex aliases, operators 212, 213
count-distinct function 210
count function 209
functions, applying on fields 209
lowercase functions 210
parts 201
SQL equivalent, viewing 204, 205
summary views, grouping for 211
uppercase functions 210

W

webapp e-commerce
 exploring 28-32
webapp order
 exploring 32-34
webapp order, contact information editing
 database, updating 40
 database entity, changing 39

entity definition, editing 39, 40
field, adding 37, 38
FTL file, creating 44, 45
modifications, checking 41, 42
new wiring, checking 45
preprocessing, creating 46, 47
save button, rewiring 42, 43
structure, changing 40, 45
user interface, editing 41, 45
widget screen, creating 43-45
wiring, creating for preprocessing 45, 46

wiring, testing 47
webapps, OFBiz
about 60
creating 61, 62
equipping, screen widget view handler
 used 69-71
testing 63
URIs 62
web server (catalina) work files
backing up 51
removing 49
restoring 52



Thank you for buying Apache OFBiz Development

Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing Apache OFBiz Development, Packt will have given some of the money received to the Apache Open For Business project.

In the long term, we see ourselves and you – customers and readers of our books – as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

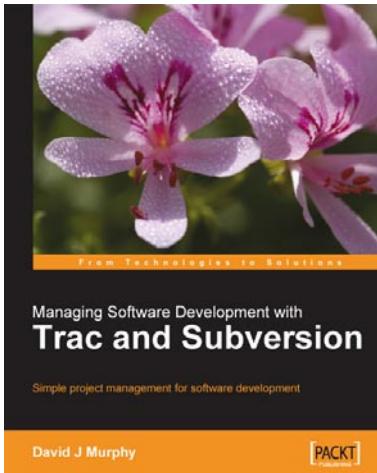
We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.PacktPub.com.

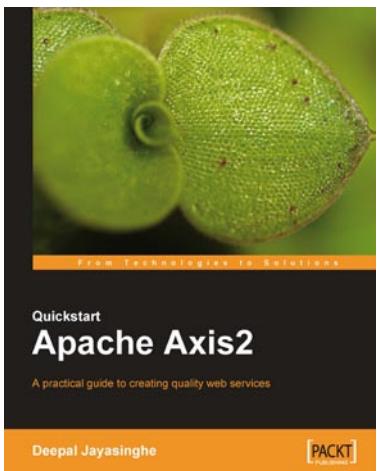


Managing Software Development with Trac and Subversion

ISBN: 978-1-847191-66-3 Paperback: 120 pages

Simple project management for software development.

1. Managing software development projects simply
2. Configuring a project management server
3. Installing, configuring, and using Trac
4. Installing and using Subversion



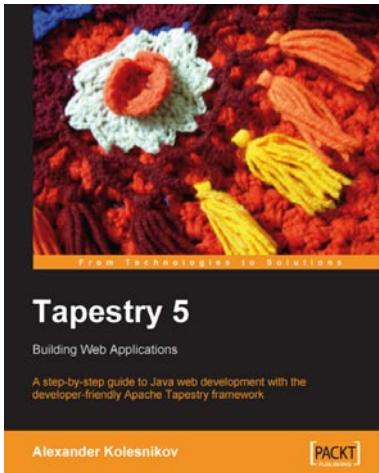
Quickstart Apache Axis2

ISBN: 978-1-847192-86-8 Paperback: 168 pages

A practical guide to creating quality web services

1. Complete practical guide to Apache Axis 2
2. Using Apache Axis2 to create secure, reliable web services quickly
3. Write Axis2 modules to enhance web services' security, reliability, robustness and transaction support

Please check www.PacktPub.com for information on our titles



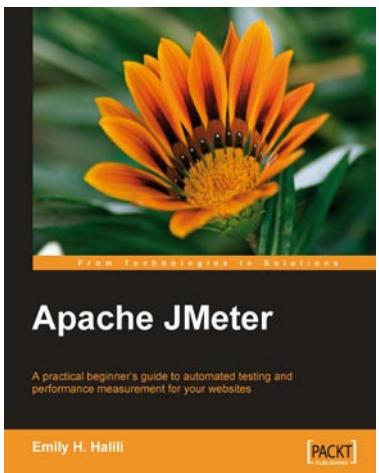
Tapestry 5

ISBN: 978-1-847193-07-0

Paperback: 280 pages

A step-by-step guide to Java Web development with the developer-friendly Apache Tapestry framework

1. Latest version of Tapestry web development framework
2. Get working with Tapestry components
3. Gain hands-on experience developing an example site
4. Practical step-by-step tutorial



Apache JMeter

ISBN: 978-1-847192-95-0

Paperback: 140 pages

A practical beginner's guide to automated testing and performance measurement for your website

1. Test your website and measure its performance
2. Master the JMeter environment and learn all its features
3. Build test plan for measuring the performance
4. Step-by-step instructions and careful explanations

Please check www.PacktPub.com for information on our titles