

Developing rich Internet applications with Rails, OpenLaszlo, and Eclipse

Automate quickly with Rails

Skill Level: Intermediate

[Robi Sen \(robisen@gmail.com\)](mailto:robisen@gmail.com)

Freelance Writer

Department13

12 May 2006

Explore at a high level how to develop a rich Internet application using OpenLaszlo, Ruby on Rails, MySQL, and Eclipse to provide a common IDE to not only develop your application but also to automate many of the steps in developing a Rails or OpenLaszlo application. This will further speed up and streamline the already fast development cycle of Rails applications.

Section 1. Before you start

Ruby on Rails has over the past year become one of the fastest growing and most popular open source Web application development frameworks. But because of the focus on building HTML applications, some have criticized Rails as being inflexible. Especially with the emergence of rich Internet applications (RIAs), applications using technologies such as Flash for user interface development and XML for data transport to replicate desktop application functionality, open source developers have wondered if there is some way to easily create RIAs that can exploit the pure object-oriented language of Ruby and the unique rapid application development features of Rails.

About this tutorial

This tutorial will expose you to two of the hottest platforms in Web development: Ruby on Rails, for fun and rapid Web application development; and OpenLaszlo, the open source RIA server. You will see that creating visually appealing desktop-like applications deployed and managed over the Web can be incredibly easy with these platforms.

While this tutorial will not teach you Ruby or Laszlo, it will show how easy it is to use Rails with RIAs using the open source OpenLaszlo Presentation server and MySQL. It will also cover how to configure Eclipse to set up a Rails and Laszlo development environment, automate mundane tasks, and create a simple Laszlo client that interacts with Rails via a Representational State Transfer (REST) Web service.

Prerequisites

To get the most from this, you should have a basic understanding of programming, XML, and SQL.

System requirements

You will need the following tools:

- [Eclipse Web Tools Platform \(WTP\) all-in-one bundle](#) or Eclipse V3.1/3.2 with WTP and all necessary extensions
- [OpenLaszlo](#)
- [MySQL V5.0](#)
- [Ruby V1.8.4](#) (one-click installer)
- [Rails V1.0](#) (note that there are issues with the latest Ruby and Rails V1.1)
- [Java technology V1.4 or 1.5](#)
- [Subversion client](#)

A system with at least 512 MB of RAM is also required.

Section 2. Setting up

This tutorial makes use of several applications and tools that require a rather lengthy setup process. Once finished, you will have the perfect open source environment for

building RIAs. In this section, you will install MySQL, OpenLaszlo Presentation Server, Ruby, Rails, the Eclipse WTP all-in-one bundle, Ruby Development Tools (RDT), and the IDEforLaszlo.

Installing MySQL

You will use MySQL for the database. Follow these steps to install it:

1. Download MySQL (see [System requirements](#)). At the time of writing, the latest version is V5.0.
2. Double-click on the installer.
3. Accept all the defaults, making sure to note the login and password you use for MySQL accounts.
4. When prompted, select Execute and MySQL will be installed.

Now that MySQL is set up and installed, let's set up OpenLaszlo V3.2.

Installing OpenLaszlo V3.2

OpenLaszlo can be used on multiple operating systems, but for this tutorial, you will use the installer for Windows® (see [System requirements](#) for download information). Once you have downloaded the installer:

1. Double-click `openlaszlo-3.2-windows-dev-install.exe`.
2. Read and accept the license agreement by clicking **I Agree**.
3. Select a directory to install the OpenLaszlo server (you will use `C:\OpenLaszloServer3.2`), then click **Install**.
4. Click **Finish**.

Depending on your system, Laszlo may take a while to install. After Laszlo finishes installing, it will start the Laszlo Presentation server and launch your default presentation server. You should see something like what's shown in Figure 1.

Figure 1. The OpenLaszlo server



The OpenLaszlo explorer contains a wealth of information, besides just documentation, such as demo code and an interactive tutorial.

There is one final step necessary for making the IDEforLaszlo work with Eclipse. Navigate to `C:\whereyouinstalledlaszlo\Server\lps-3.2\WEB-INF\lib` -- for example, `C:\OpenLaszloServer3.2\Server\lps-3.2\WEB-INF\lib` -- and create a test file called `makelaszlowork3.1.txt`. Without this file, you will not be able to make the IDEforLaszlo plug-in work.

Note: The plug-in is hardcoded to look for a specific context root of 3.1, so using the latest version of Laszlo will cause problems if you don't add this file.

Installing Eclipse WTP

The Eclipse WTP project focuses on extending the Eclipse platform for building Web applications (Java™ 2 Enterprise Edition (J2EE) specifically, but with little effort, it

can be used for almost any Web application platform). The WTP depends on a number of other Eclipse projects -- such as the Eclipse Modeling Framework (EMF), and the Graphical Editing Framework (GEF). So for this example, start by downloading the WTP all-in-one bundle that includes all these dependencies, as well as Eclipse, in one easy-to-use bundle. You can download the WTP, as well as the associated dependencies, and install them, but this will not be covered in this tutorial.

Note: If you have Eclipse installed, but want to try installing the WTP all-in-one bundle, as well, you can. Although it is not recommended for this tutorial, you can have both versions share the same workspace, allowing you to see past projects created using other versions of Eclipse.

To set up the WTP all-in-one bundle:

1. Download wtp-all-in-one-sdk-1.0-win32.zip (see [System requirements](#)).
2. Unzip wtp-all-in-one-sdk-1.0-win32.zip into a directory -- C:\software\eclipseWTP, for example.
3. You should see eclipse.exe in the directory you unzipped the bundle. You can start the Eclipse IDE by double-clicking on the .exe -- or better yet, right-click on the .exe, drag it to your desktop, and release the right mouse button. A menu should pop up asking if you would like to create a shortcut. Do that. From here, you can double-click on the shortcut to start the IDE. Do that now.
4. Eclipse will start and prompt you to select a workspace. The default is usually something like c:\Documents and Settings\userid\workspace. For now, select **Default**. You can change your workspace later.
5. You should now see something like that shown in Figure 2.
Figure 2. The Eclipse welcome screen the first time it is launched



From here, select the large bent arrow in the upper right-hand corner. When you mouse over the arrow, it will show **To Workbench**. Click on it, and you will be taken to the default view of Eclipse IDE V3.1.

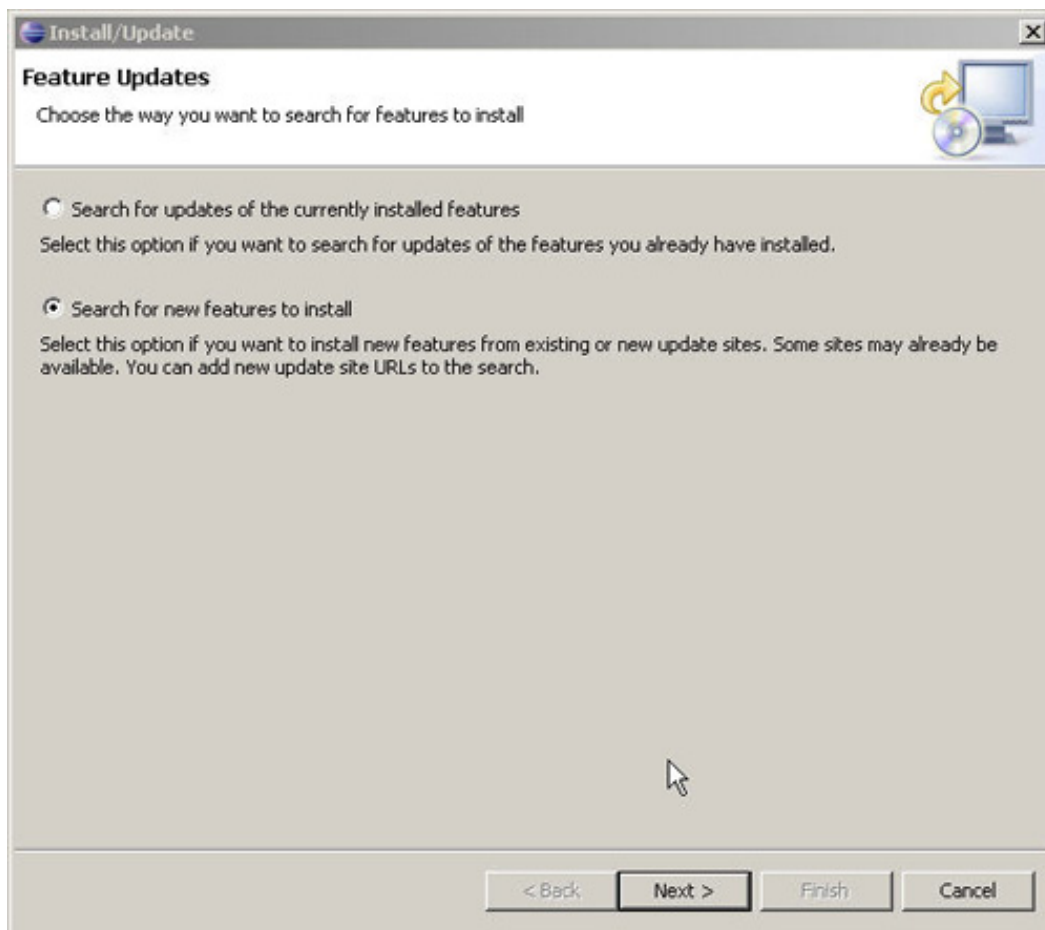
Next, you need to set up Eclipse to work with Laszlo.

IDE for Laszlo

OpenLaszlo has its own plug-in for Eclipse, letting you easily edit, update, visually design using drag-and-drop components, and even preview your Laszlo applications. To install the IDEforLaszlo:

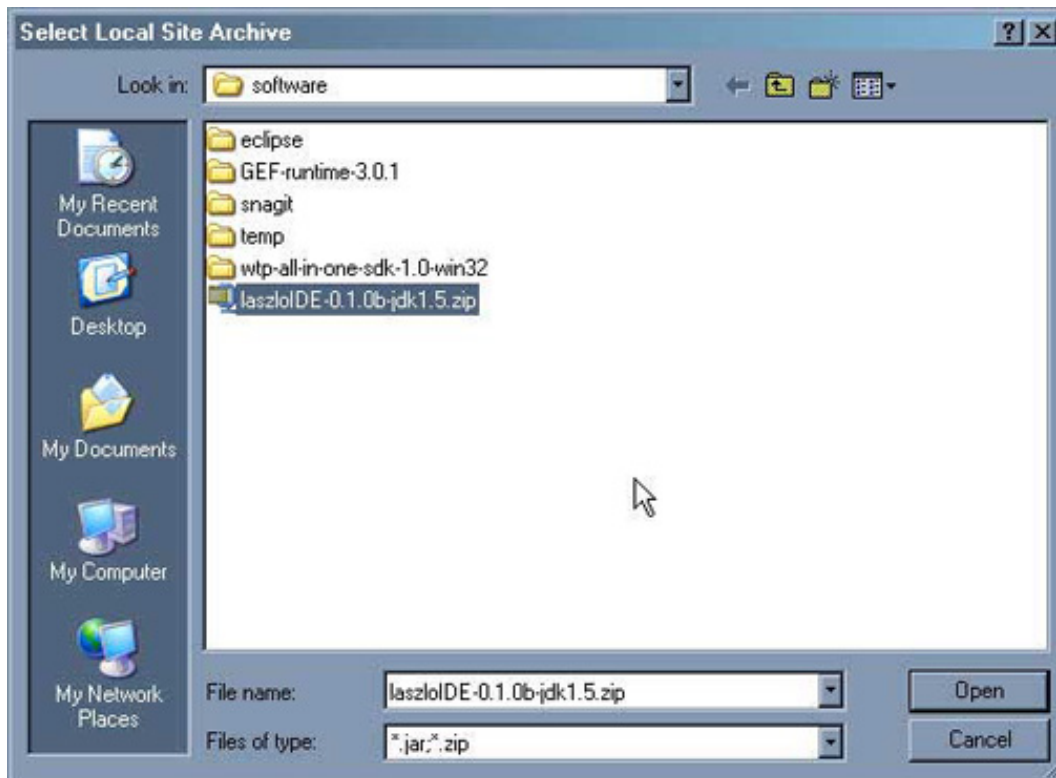
1. Download the latest IDE version as a zip (see [System requirements](#)), depending on the version of Java you have on your system (at the time of writing it was laszloIDE-0.1.0b-jdk1.5.zip).
2. Start your Eclipse WTP.
3. When the Eclipse IDE starts up go to: Select **Help > Software Updates > Find and Install**.
4. On the Feature Updates page, select Search for new features to install, and you should see something like that shown in Figure 3. Then click **Next**.

Figure 3. Install/Update wizard for installing new Eclipse plug-ins



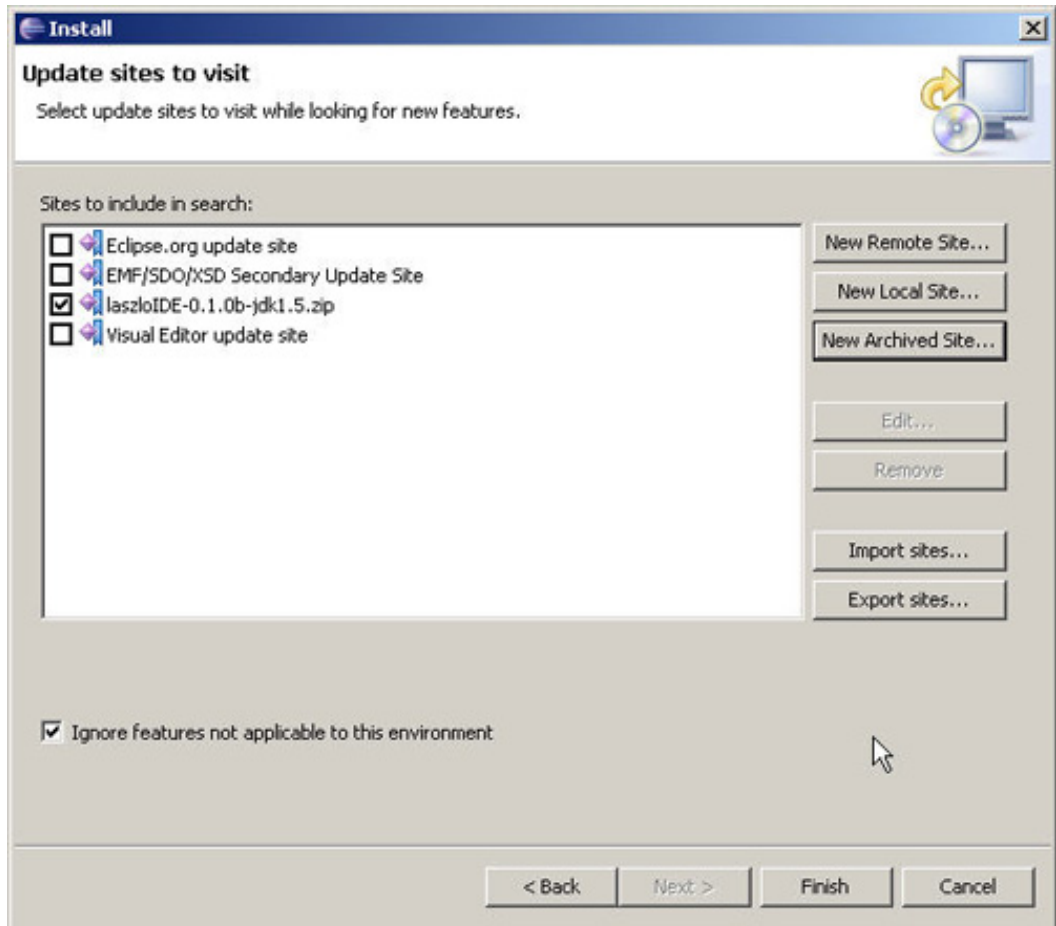
5. On the Update sites to visit page, click **New Archived Site**.
6. Browse and find the laszloIDE-0.1.0b-jdk1.5.zip file, then click **OK**. You should see something like that shown in Figure 4.

Figure 4. The LaszloIDE archive selected for installation



7. On the Edit Local Site dialog, confirm the zip location by clicking **OK**.
8. Back on the Update sites to visit page (see Figure 5), with the laszloIDE-0.1ob-jdk1.5.zip checked, click **Finish**.

Figure 5. Check LaszloIDE archive to make sure it is installed



9. On the Search Results page, check laszloIDE-0.1ob-jdk1.5.zip, then click **Next**.
10. On the Feature License page, read and accept the license, then click **Next**.
11. On the Installation page, click **Finished**.
12. On the Feature Verification page, click **Install All**.
13. When prompted to restart, click **Yes**.

Once it restarts, it should start up in the Laszlo view. At this point, you just need to set up Ruby and Rails, and get Eclipse customized for Rails, then you can create your Laszlo on Rails application.

Setting up Ruby and Rails

For Windows users, the easiest way to set up Ruby is to download the one-click installer (see [System requirements](#)). You can download the latest .exe (at the time of writing, it was ruby184-16p3.exe).

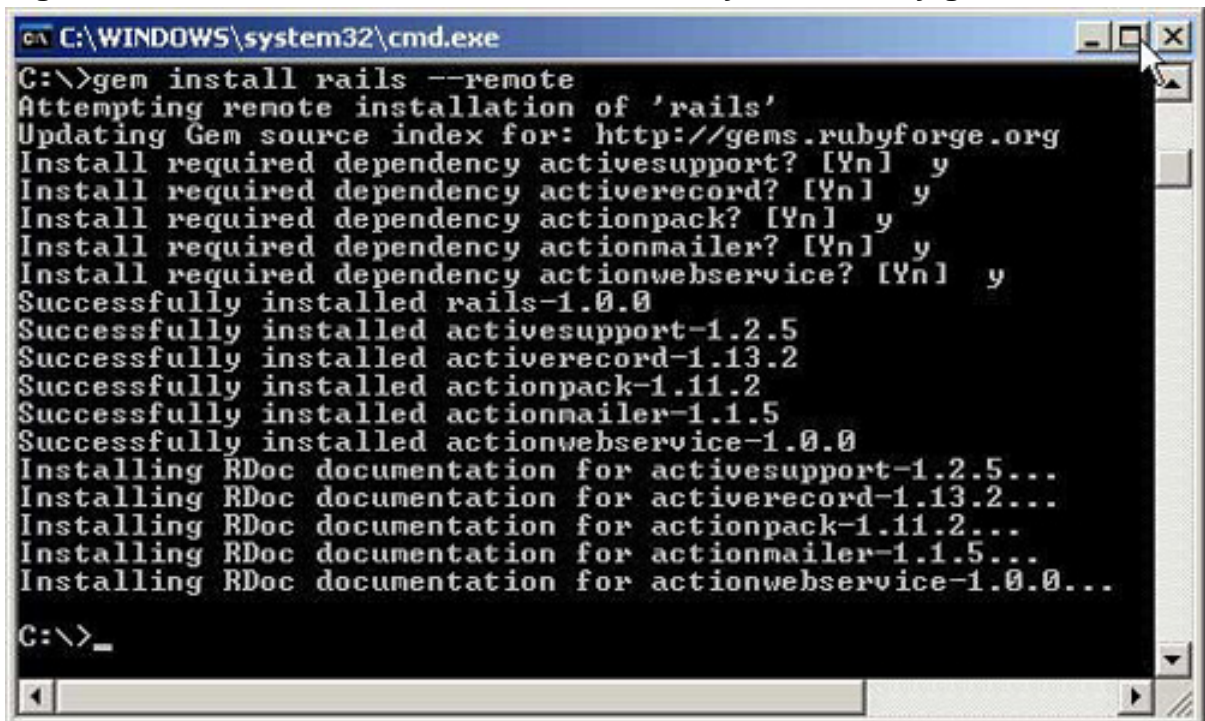
Note: You can also download Ruby or its source from Ruby-lang.org, but the one-click installer makes your life easier by including Ruby and a number of other useful tools.

Once you have downloaded the one-click Ruby Installer:

1. Double-click on the installer. When prompted, select a directory to install to. For this tutorial, use C:\Ruby, but you can select anywhere on your drive.
2. Once Ruby is installed, go to your command prompt via **Start > Run**, then typing `cmd`.
3. Install Rails by typing in the command prompt: `gem install rails --include-dependencies`.

This will start downloading Rails and begin installation. You will be prompted multiple times to install required dependencies. Type `y` at each prompt, and you should see something like Figure 6.

Figure 6. Results from the command line when you use Ruby `gem install rails`



```
C:\WINDOWS\system32\cmd.exe
C:\>gem install rails --remote
Attempting remote installation of 'rails'
Updating Gem source index for: http://gems.rubyforge.org
Install required dependency activesupport? [Yn] y
Install required dependency activerecord? [Yn] y
Install required dependency actionpack? [Yn] y
Install required dependency actionmailer? [Yn] y
Install required dependency actionwebservice? [Yn] y
Successfully installed rails-1.0.0
Successfully installed activesupport-1.2.5
Successfully installed activerecord-1.13.2
Successfully installed actionpack-1.11.2
Successfully installed actionmailer-1.1.5
Successfully installed actionwebservice-1.0.0
Installing RDoc documentation for activesupport-1.2.5...
Installing RDoc documentation for activerecord-1.13.2...
Installing RDoc documentation for actionpack-1.11.2...
Installing RDoc documentation for actionmailer-1.1.5...
Installing RDoc documentation for actionwebservice-1.0.0...
C:\>_
```

Rails should now be installed. Let's get Eclipse set up so you can edit Ruby files and automate some of the command-line chores usually associated with Rails projects.

Setting up RDT on Eclipse

The RDT plug-in gives Eclipse users the standard interface and tools they expect for developing code, except that it is targeted at Ruby users. In this section, you will learn how to install the RDT plug-in and customize your Eclipse environment to take some of the drudgery out of the command-line tasks usually required when developing Ruby on Rails applications using a standard text editor. So, to get the RDT plug-in installed on your version of Eclipse:

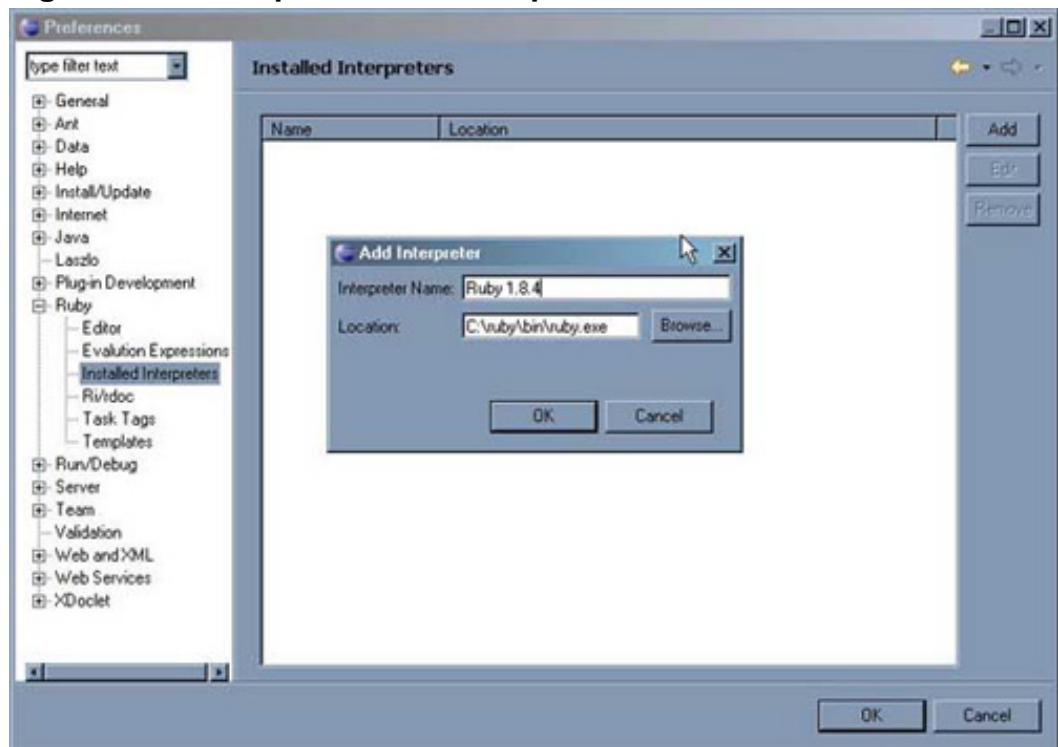
1. Ensure that Eclipse is running.
2. Go to **Help > Software Updates > Find and Install**.
3. Select **Search for new features to install** and click **Next**.
4. Go to New Remote Site.
5. For the site name, use `Ruby Development Tools`.
6. For the URL, use `http://updatesite.rubypeople.org/release`.
7. Click **OK**.
8. Check the box for RDT and click **Next**.
9. Select the feature RDT and click **Next**.
10. Confirm any messages relating to installing unsigned plug-ins.
11. RDT should install, then you'll be prompted to restart Eclipse. Select **OK**.

Now you want to configure the environment for doing Ruby and Rails in Eclipse. Start Eclipse, then:

1. Go to **Window > Preferences**, and from the Preferences tree, expand Ruby.
2. Select Installed Interpreters.
3. Click **Add**.
4. Enter `Ruby` as the interpreter name.

5. Enter the path to your ruby.exe file (usually, C:\ruby\bin\rubyw.exe). You should see something like what's shown in Figure 7.

Figure 7. The Eclipse Preferences pane



6. Select **OK**.
7. Then select **OK** to close the Preferences pane.

Now that you have set up Ruby, you want to use Eclipse to make working with Rails easier. You are going to set up Eclipse so it will call external Rails scripts without you having to go to the command line to type them manually. If you are a regular Rails user or are interested in using Rails with Eclipse, make note of the technique used here, which can also be used for running regular Rails script operations.

In the next section, you will use Eclipse's ability to pass commands to the Windows command line to automate these tasks.

Section 3. Launching external Rails scripts with Eclipse

Rails provides a number of scripts that automate much of the work in setting up a

Rails project. The first of these is the Rails command, which automatically creates a file system for your Rails projects. There are numerous other commands that allow you to do things from starting the internal Web server to running build scripts to generating functional code to creating skeletons of applications (*scaffolds*). These are usually run from the command line, which can be annoying if you have to keep moving from your IDE to the command line and back. In this section, you'll learn how to use Eclipse to launch these scripts from inside the Eclipse environment. You'll use the same technique several times, and the first script you will automate is the script for building a Rails project.

Note: You should know that these settings are tied to your workspace. If you change workspaces, you will need to reconfigure these options. It is hoped that the fine developers of RDT will build this ability into the plug-in at some point so they appear on the context menus.

Setting up Eclipse to create a Rails project

To start, open Eclipse to the Ruby perspective, then:

1. Go to **Window > Open Perspective > Other**, then select Ruby.
2. Select **Run > External Tools > External Tools**.
3. Select **Program** from the configuration tree.
4. Select **New** to run a new program.
5. For the name, enter `Create Rails Project`.
6. For the Location, enter `C:\ruby\bin\rails.cmd`.
7. For the Working Directory, enter `${project_loc}/${project_name}`.
8. For Arguments, enter `{project_name}`.
9. Go to the Refresh tab and select it. Check **Refresh Resources Upon Completion**.
10. Select the Common tab, find the section for Display in Favorites Menu, and check **External Tools**.
11. **Apply**.

When you select Create Rails Project from External Tools, it now will automatically

build a Rails project in your project directory.

Let's set up Eclipse to download a Rails plug-in called simply OpenLaszlo Rails Plug-in. The OpenLaszlo Rails Plug-in, developed by Oliver Steele, allows you to easily set up and build OpenLaszlo clients by supplying generators and scaffolds.

Setting up Eclipse to install the OpenLaszlo Rails plug-in

The OpenLaszlo Rails plug-in needs to be called once for each Rails application in which you want to use it. The script downloads the plug-in and installs it in your Rails project. For this step to work, you must have a Subversion client installed. Follow these instructions:

1. Select **Run > External Tools > External Tools**.
2. Select **Program** from the configuration tree.
3. Select **New** to create a new program.
4. For Name, use `Setup Openlaszlo plug-in`.
5. For Location, enter `C:\ruby\bin\ruby.exe`.
6. For Working Directory, use `${project_loc}/${project_name}`.
7. For Arguments, use `./script/plugin install \`
`svn://rubyforge.org/var/svn/laszlo-plugin/tags/openlaszlo`.
8. Go to the Refresh tab and check **Refresh Resources Upon Completion**.
9. Go to the Common tab, find the section for Display in Favorites Menu, and check **External Tools**.
10. **Apply**.
11. To install the OpenLaszlo plug-in, instead of going to the command line and typing something like:

```
>Ruby ./script/plugin install \  
svn://rubyforge.org/var/svn/laszlo-plugin/tags/openlaszlo
```

12. Select the Setup OpenLaszlo plug-in from External Tools.

Let's set up another external script call for creating REST controllers.

Setting up Eclipse to generate REST controllers

You want to be able to have Ruby use the OpenLaszlo plug-in to automatically create the REST controller. REST is a common architectural method of creating simple XML-based Web services. OpenLaszlo uses XML to communicate with Rails, and Rails needs to be able to accept XML from OpenLaszlo, as well as pass XML. In most application server environments, developing a REST XML API can be a pain, but with the OpenLaszlo plug-in, it's as simple as calling a single script.

Open Eclipse, then:

1. Select **Run > External Tools > External Tools**.
2. Select **Program** from the configuration tree.
3. Select **New** to create a new program.
4. For Name, use `Generate REST Controllers`.
5. For Location, use `C:\ruby\bin\ruby.exe`.
6. For Working Directory, use `${project_loc}/${project_name}`.
7. For Arguments, use `./script/generate rest_controller ${project_name}`.
8. Go to the Refresh tab and check **Refresh Resources Upon Completion**.
9. Go to the Common tab, find the section for Display in Favorites Menu, check **External Tools**, then select **Apply**.

Whenever you need to, you can now have OpenLaszlo create a REST controller at the click of button.

Next, you want to have Ruby use the OpenLaszlo plug-in to generate a skeleton or scaffold OpenLaszlo application.

Setting up Eclipse to create Laszlo applets

One cool thing about Rails is that it automates much of the drudgery of Web application development. One of the most demonstrated features of Rails is its ability to create scaffolds -- or simple Create, Retrieve, Update, and Delete (CRUD) HTML interfaces -- that map to the database, allowing you to quickly get an interface up, and from there, customize the applications presentation. This provides the scaffold

from which your application is built.

The OpenLaszlo plug-in allows much the same thing. By calling the OpenLaszlo script/generate applet, OpenLaszlo can create a scaffold client you can use for your application. To automate this to be called from Eclipse, instead of the command line, open Eclipse, then:

1. Select **Run > External Tools > External Tools**.
2. Select **Program** from the configuration tree.
3. Select **New** to create a new program.
4. For Name, use `Create Laszlo Applets`.
5. For Location, use `C:\ruby\bin\ruby.exe`.
6. For Working Directory, use `${project_loc}/${project_name}`.
7. For Arguments, use `./script/generate applet
${string_prompt:Applet name} applet`.
8. Go to the Refresh tab and check **Refresh Resources Upon Completion**.
9. Go to the Common tab, find the section for Display in Favorites Menu, and check **External Tools**.
10. **Apply**.

The last external tool to set up allows you to start Rails' internal Web server you will use for development. The Web server is called WEBrick. In the next section, you will set this up as an external tool, as well.

Calling Rails WEBrick from Eclipse

Rails comes with its own simple HTTP server, created for building and testing your applications. You usually call Rails from within your Rails application directory like `Ruby script/server`, but you want to try to stay in the Eclipse environment as much as you can, allowing you to use just one tool so you can focus on developing applications. To set up Eclipse to call WEBrick, open Eclipse, then:

1. Select **Run > External Tools > External Tools**.
2. Select **Program** from the configuration tree.

3. Select **New** to create a new program.
4. For Name, use `Start WEBrick`.
5. For Location, use `C:\ruby\bin\ruby.exe`.
6. For the Working Directory, use `${project_loc}/${project_name}`.
7. For Arguments, use `script\server`.
8. Go to the Refresh tab and check **Refresh Resources Upon Completion**.
9. Go to the Common tab, find the section for Display in Favorites Menu, and check **External Tools**.
10. **Apply**.

So now, when you test your Rails application, you can simply start up and shut down the Rails development server from Eclipse.

Note: WEBrick is a Rails development Web server and should not be used for production. Consult the Ruby on Rails Web site for more information about setting up Rails for use with your specific production Web server (see [Resources](#)).

Tip: For developers already using RTD or who plan to do more typical Rails applications, you can automate almost any of the command-line calls you normally use (such as generate scaffolds or models) using the above technique. Try it and you will quickly see that you can use Eclipse to radically increase your development time. No more typing on the command line!

Section 4. Creating a simple movies list

It probably seemed like you spent the whole tutorial setting up Eclipse, instead of developing an application, but that's the point. As you will quickly see, when you start to create the simple rich Internet application in the remainder of the tutorial, almost no effort is required to quickly create a scaffold or skeleton application.

Creating the database

You need to create a database for your application. Most Rails applications start with

the development of a database. One of Ruby on Rails' interesting approaches is the idea of "convention over configuration," which essentially means that Rails forces certain naming conventions on you, allowing Rails to use reflection and discovery to ascertain almost everything it needs to know about your application. Rails can then automate much of the drudgery of Web development, as well as remove the need for complex XML configuration files. Let's build the database, and you will soon see what I mean.

Tip: You can also use Eclipse to interact with and develop databases (see [Resources](#)).

For your database, you're going to make a simple table to create a list of your favorite movies and allow you to rate them. To do this, you will use the MySQL command-line client, but if you are more comfortable with a specific MySQL client, by all means, use it.

Build the database

To create the database:

1. Go to to **Start > MySQL > MySQL 5.0 > MySQL Command Line**.
2. Log in.
3. At the prompt, type `create database movies;`, then **Enter**.
4. At the prompt, type `grant all on movies.* yourlogin@localhost;`, then **Enter**.
5. Use this SQL to create the movies table:
Listing 1. SQL to create the movie table

```
CREATE TABLE Movies (  
  id MEDIUMINT NOT NULL AUTO_INCREMENT,  
  movie_name CHAR(30) NOT NULL,  
  movie_genre CHAR(30) NOT NULL,  
  movie_description TEXT NOT NULL,  
  movie_rating MEDIUMINT,  
  PRIMARY KEY (id)  
);
```

Let's create a Ruby on Rails project and start developing an application.

Section 5. Creating the Rails project

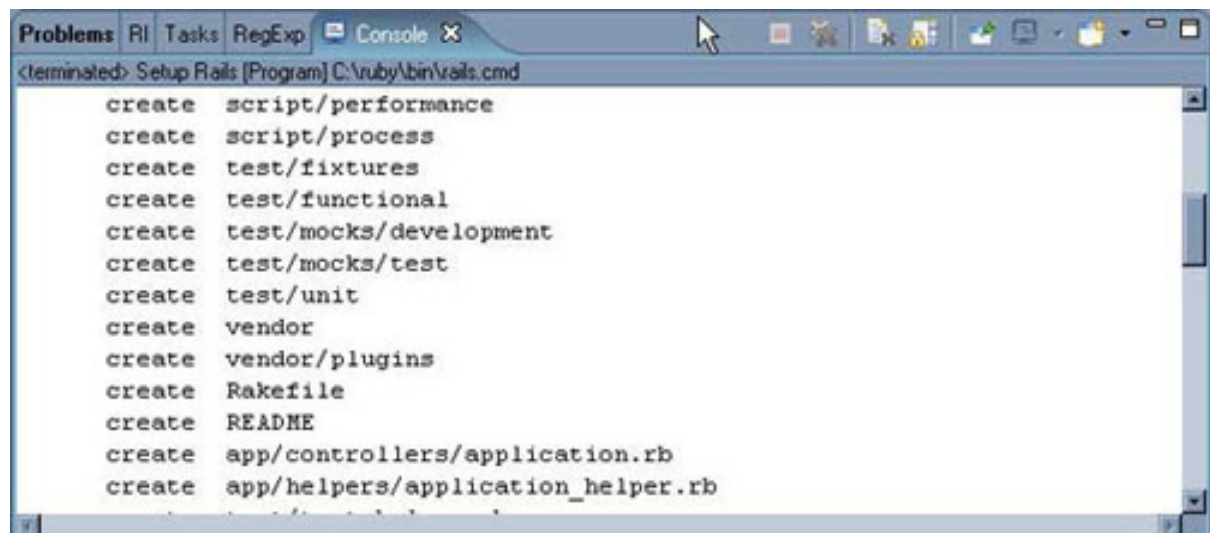
It's time to create a Ruby on Rails project with Eclipse. Make sure you have your Eclipse editor running.

Creating the movie project

Create the new project by doing the following:

1. Make sure you are in the Ruby perspective. If you are not, go to **Windows > Open Perspective > Other**, then select Ruby.
2. Go to **File > New > Project > Ruby**.
3. Enter the name of the project as `movies` and use the default workspace. You should then see a folder called `movies` in your Ruby Resources pane.
4. Go to **Run > External Tools > Setup Rails**. As soon as you do this, you will see a pop-up progress bar. When it's done, you should see something in the Eclipse Console like Figure 8.

Figure 8. Example output in Eclipse console when External Tools Setup Rails is run



```
<terminated> Setup Rails [Program] C:\ruby\bin\rails.cmd
create  script/performance
create  script/process
create  test/fixtures
create  test/functional
create  test/mocks/development
create  test/mocks/test
create  test/unit
create  vendor
create  vendor/plugins
create  Rakefile
create  README
create  app/controllers/application.rb
create  app/helpers/application_helper.rb
```

Look in your Ruby Resources pane, and you should see a bunch of directories. When you run the Rails external tool, it not only builds all of these directories but it

also creates some files.

Telling Rails what database to use

One of the configuration files to set up in Rails is the `database.yml` file that tells Rails what database to use for development, testing, or production. If you look in your Rails project from Eclipse, you should see a directory called `config`. Once you have found it:

1. Expand the `config` folder and double-click on the `database.yml` file.
2. You should see a file containing a bunch of database configuration information. Edit it as shown in Listing 2, with the exception of entering in your database user name and password.

Listing 2. Configuring the `database.yml` file

```
development:
  adapter: mysql
  database: movies
  username: yourusername
  password: yourpassword
  socket: /path/to/your/mysql.sock

test:
  adapter: mysql
  database: movies
  username: yourusername
  password: yourpassword
  socket: /path/to/your/mysql.sock

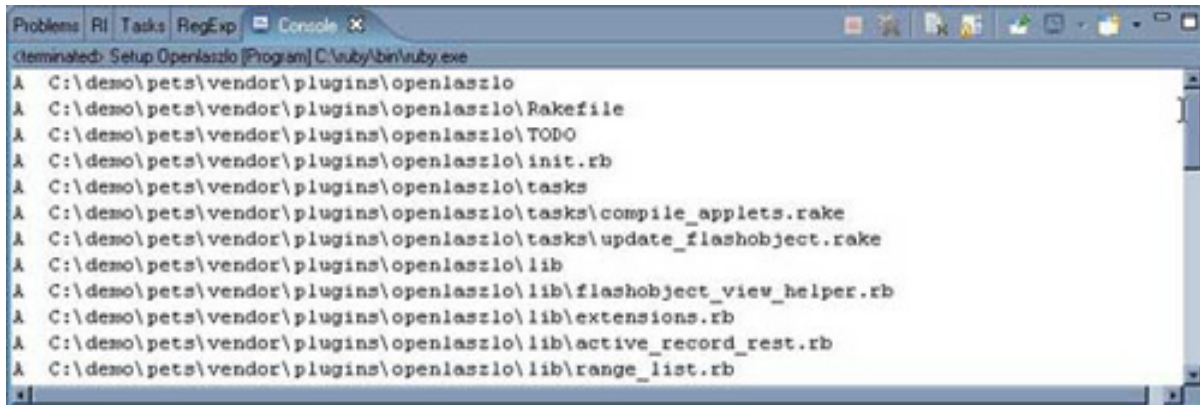
production:
  adapter: mysql
  database: movies
  username: yourusername
  password: yourpassword
  socket: /path/to/your/mysql.sock
```

3. Go to **File > Save** or just use **Ctrl+S** to save your file.

Installing the OpenLaszlo Rails plug-in

You need to install OpenLaszlo into your application directory now. Go to **Run > External Tools > Setup OpenLaszlo**. You should see something like what's shown in Figure 9.

Figure 9. Output from command line as OpenLaszlo plug-in is installed in Rails application directory



```

(terminated: Setup OpenLaszlo [Program] C:\ruby\bin\ruby.exe
A C:\demo\pets\vendor\plugins\openlaszlo
A C:\demo\pets\vendor\plugins\openlaszlo\Rakefile
A C:\demo\pets\vendor\plugins\openlaszlo\TODO
A C:\demo\pets\vendor\plugins\openlaszlo\init.rb
A C:\demo\pets\vendor\plugins\openlaszlo\tasks
A C:\demo\pets\vendor\plugins\openlaszlo\tasks\compile_applets.rake
A C:\demo\pets\vendor\plugins\openlaszlo\tasks\update_flashobject.rake
A C:\demo\pets\vendor\plugins\openlaszlo\lib
A C:\demo\pets\vendor\plugins\openlaszlo\lib\flashobject_view_helper.rb
A C:\demo\pets\vendor\plugins\openlaszlo\lib\extensions.rb
A C:\demo\pets\vendor\plugins\openlaszlo\lib\active_record_rest.rb
A C:\demo\pets\vendor\plugins\openlaszlo\lib\range_list.rb

```

You can now use the OpenLaszlo plug-in to automatically create REST XML interfaces for the simple application.

Once again, go to **Run > External Tools > Generate REST Controllers**. Watch your Eclipse console. You should see the OpenLaszlo plug-in creating a bunch of files, including your Rails Model and a Rails Controller.

Looking at the movie model

Let's look at the model for the controller by navigating to **apps > models >** and double-clicking on `movie.rb`. You should see something like:

```
class Movie < ActiveRecord::Base
end
```

These two lines of code demonstrate the power of Rails. The line `class Movie` actually creates a `Movie` object that Rails can then map to the `movies` table in the database. Rails can do this because of the specific naming convention it uses for databases, which allows it to map models that use a singular naming convention -- in this case, `Movie` -- to tables always named in the plural form.

Rails also creates methods for accessing rows and attributes in your tables. Rails does this by implementing the ActiveRecord pattern, which allows Rails to create a class that knows everything it needs to know about interacting with the database.

Tip: Read more about how ActiveRecord and database wrappers work by reading "Crossing borders: Exploring Active Record" (see [Resources](#)).

While you might think this is a limitation, there are very few cases where you can not create a database using these naming conventions. In those extreme cases where you are forced to use legacy database schemes, or you have some specific need for names that break this convention, they can be overridden.

Looking at the movie controller

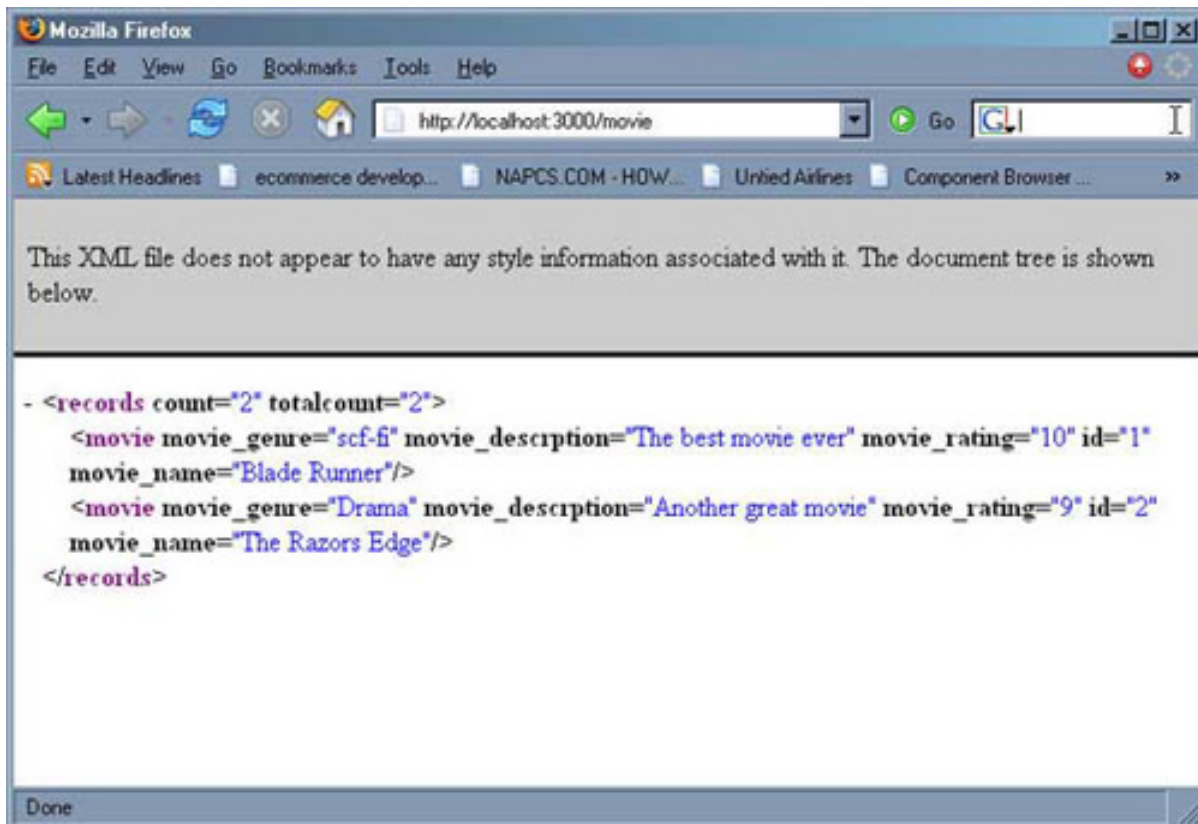
Let's look at the Rails controller for your application. Navigate to **movies > app > controllers > movie_controller.rb**, then double-click on it to open it. The code should look like what's shown in Listing 3.

Listing 3. The movie_controller.rb file

```
class MovieController < ApplicationController
  # The following line defines methods that implement the OpenLaszlo
  # REST API.
  #
  # To replace this file with a file that contains explicit method
  # definitions, execute:
  #   script/generate rest_scaffold movie
  rest_scaffold :movie
end
```

Once again, the Ruby code Rails created was just a couple lines. It's hard to imagine that a few mouse clicks have actually created a REST XML Web service (especially if you have experience using J2EE applications). Let's confirm it. Start the Rails Web server WEBrick by going to **Run > External Tools > Start WEBrick**. Launch your favorite browser and go to <http://localhost:3000/movie>. You should see something like what's shown in Figure 10.

Figure 10. An example of the XML generated by our movie controller



It is actually generating XML from the database.

Let's move on and get the Laszlo scaffold generated. First stop the WEBrick server by clicking the red **Stop** button on the Eclipse console.

Rails generates methods for you

To help understand what's going on a little bit more, let's look at the code generated by the OpenLaszlo plug-in in the case where you had used the command `script/generate rest_scaffold movie` to create a control that writes out each of the methods for the scaffold explicitly.

Listing 4. Rails creating methods on the fly

```

class MovieController < ApplicationController
  # The following line can be used instead of the generated definitions
  # of records, page, schema, create, etc. The generated definitions
  # are provided as a starting point in case you need to modify them.
  #
  #rest_scaffold :movie

  def records
    # The default route places the range list in the id parameter.
    # Retrieve it from there so that this action works with the default
    # route.

```

```

    ranges = RangeList.parse(params[:id] || '', :domain_start => 1)
    options = {}
    options[:conditions] = ranges.to_sql_condition unless ranges.empty?
    records = Movie.find :all, options
    count = Movie.count
    response.headers["Content-Type"] = "text/xml"
    render :text => RestHelper::records_xml(records, count)
  end
  alias_method :index, :records

  def page
    ranges = RangeList.parse params[:id], :domain_start => 1
    records = Movie.find_pages ranges
    count = Movie.count
    response.headers["Content-Type"] = "text/xml"
    render :text => RestHelper::records_xml(records, count)
  end
  alias_method :pages, :page

  def schema
    response.headers["Content-Type"] = "text/xml"
    render :text => RestHelper::schema_xml(Movie)
  end

  def create
    values = params['id'].split('&').map {|s| s.split('=', 2)}
    record = Movie.new(Hash[*values.flatten])
    response.headers["Content-Type"] = "text/xml"
    if record.save
      render :text => "<create>#{record.id}</create>"
    else
      render :text =>
"<error>#{record.full_messages.join("\n")}</error>"
    end
  end

  def update
    values = params['id'].split('&').map {|s| s.split('=', 2)}
    attributes = Hash[*values.flatten]
    id = attributes['id']
    attributes.delete 'id'
    record = Movie.find id
    record.attributes = attributes
    response.headers["Content-Type"] = "text/xml"
    if record.save
      render :text => "<update>#{record.id}</update>"
    else
      render :text =>
"<error>#{record.full_messages.join("\n")}</error>"
    end
  end

  def delete
    ranges = RangeList.parse params['id'], :domain_start => 1
    conditions = ranges.to_array :first_index => 1
    Movie.delete_all conditions
    response.headers["Content-Type"] = "text/xml"
    render :text => "<deleted id='#{id}'/>"
  end
end
end

```

As you can see, Rails has created a number of simple CRUD methods. Once again, Rails is smart enough to create the methods on the fly for you, but to customize these methods or create your own, you can see from this file how to go about it.

Let's create the Laszlo client to view the Rails movie model and controller.

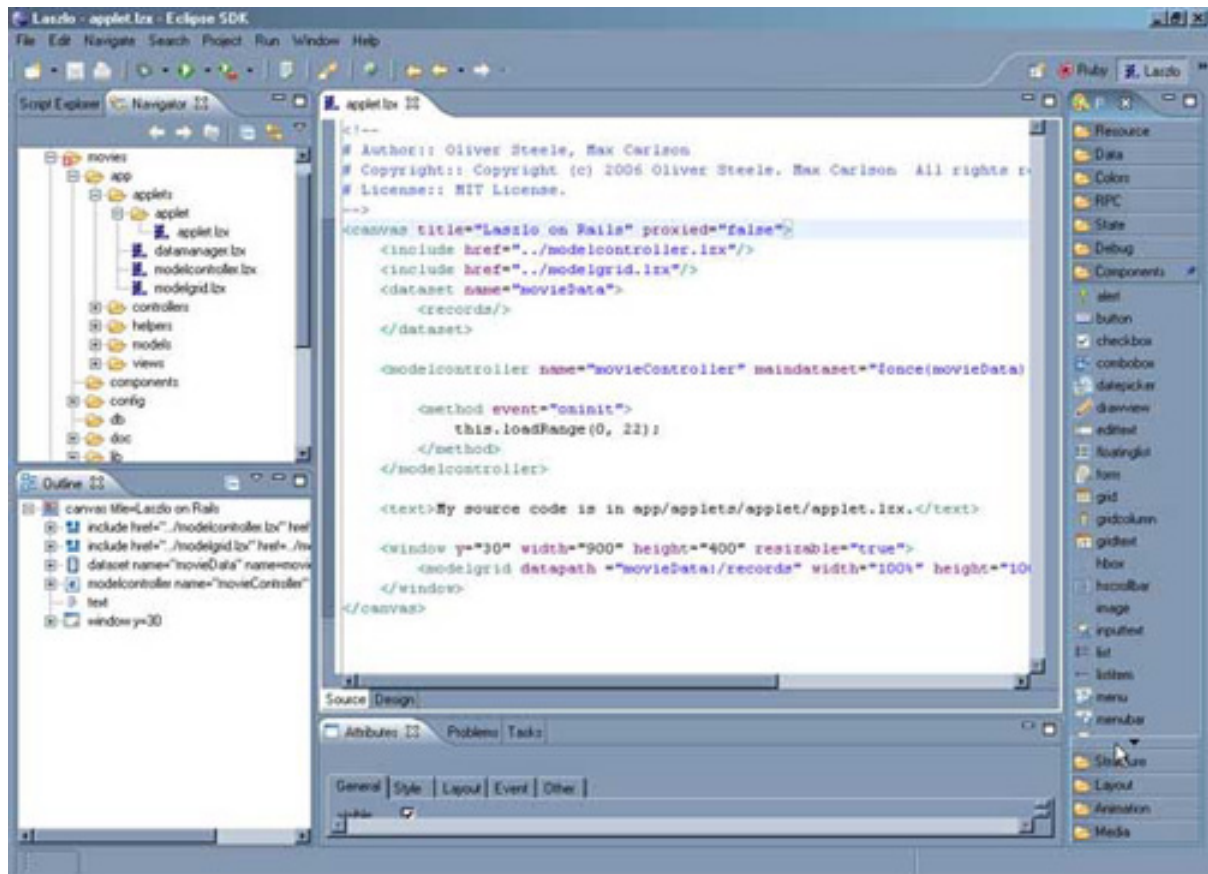
Section 6. Creating a simple Laszlo client

Generating the Laszlo scaffold

Let's create your Laszlo XML:

1. Go to **Run > External Tools > Generate Applet**, and when it prompts you for an Applet Name, enter `movie`. Note that this will not generate the Laszlo `.lzx` files that provide our view.
2. Once the generator is done creating the Laszlo view, go to the Ruby Resource pane and look for the directory `app`.
3. Expand the `app` directory and select **applets > applet**. You should see `applet.lzx` in the `applet` directory and three files: `datamanager.lzx`, `modelcontroller.lzx`, and `modelgrid.lzx`.
4. Switch to the IDEforLaszlo view by using **Window > Show View > Other**, then selecting **Laszlo**. You should see something like what's shown in Figure 11.

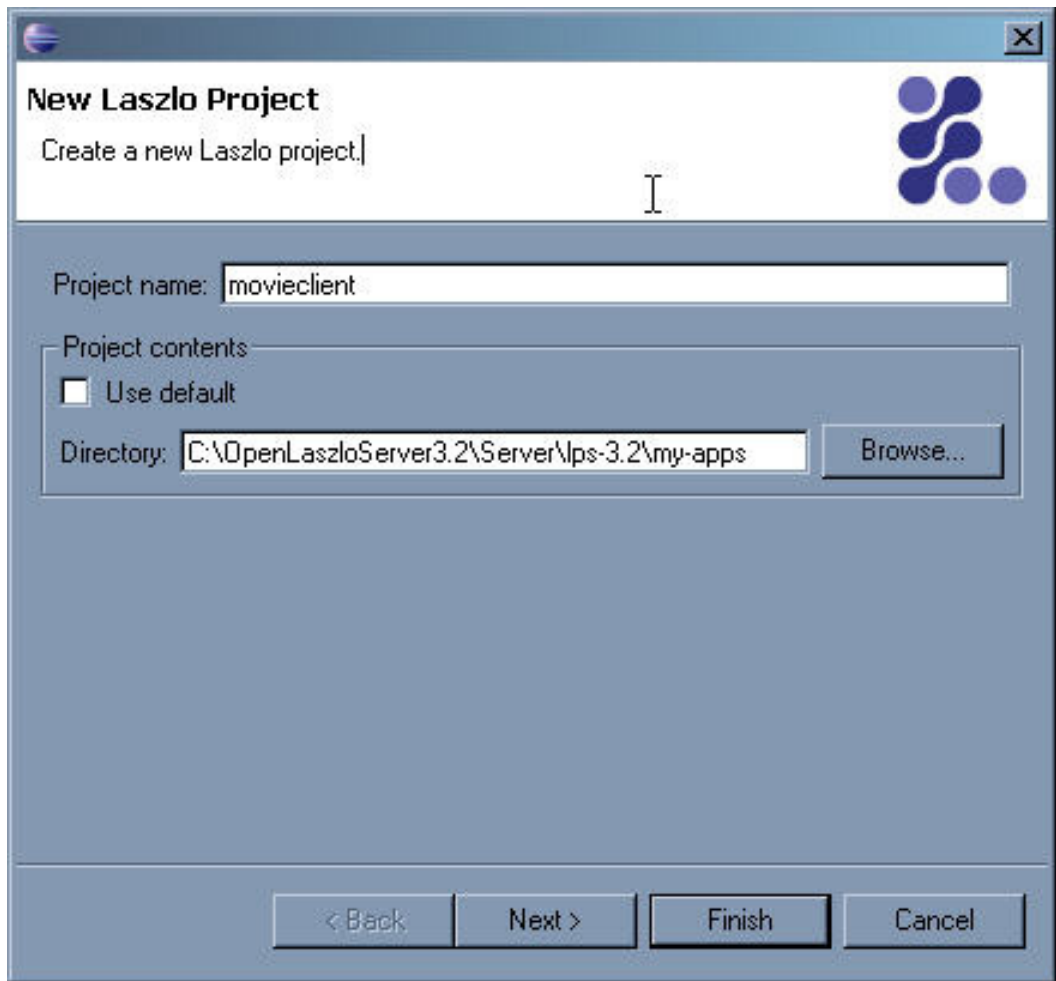
Figure 11. Example of OpenLaszlo plug-in generating `.lzx` as viewed from IDEforLaszlo view in Eclipse



Creating an OpenLaszlo project

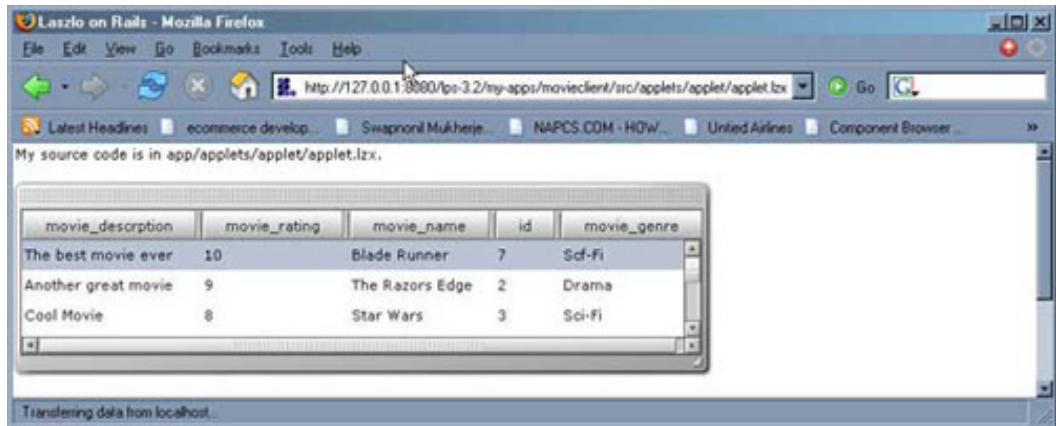
The OpenLaszlo plug-in creates a bunch of files for you. You should see under `movies\app\applets` three files: `datamanager.lzx`, `modelcontroller.lzx`, and `modelgrid.lzx`. Under `movies\app\applets\applet`, you should see `applet.lzx`. Now that you have your Laszlo code, you need to create a Laszlo project under `C:\OpenLaszloServer3.2\server\lps-3.2\my-apps` so you can run the resultant compiled Laszlo `.lzx` files on the Laszlo Presentation Server. Follow these steps to create a new Laszlo project:

1. In Eclipse, go to **File > New > Laszlo Project**.
 2. In the New Laszlo Project wizard, enter `Movie Client` for the project name, then unclick **Use default**. This will let you edit the workspace directory. Browse to or enter in the path to your OpenLaszlo Server `my apps` directory. For this tutorial, use `C:\OpenLaszloServer3.2\server\lps-3.2\my-apps`. You should see something like what's shown in Figure 12. Select **Finish**.
- Figure 12. New Laszlo Project wizard for movieclient**



3. Open your Rails project and navigate to **movie > app > applets**, then right-click on the applets folder and **Copy** (or use **Ctrl+C**).
4. Navigate to your new Laszlo project called movieclient and copy the applets folder directly into the \src folder.
5. Open a browser to <http://127.0.0.1:8080/lps-3.2/my-apps/movieclient/bin/applets/applet/>, and you should see a file called applet.swf. Click on it. You should see something like what's shown in Figure 13.

Figure 13. The Laszlo on Rails movieclient as seen from the browser



Try clicking on one of the fields, and you will notice that if you change a field and move to the next field, the database is automatically updated. You can also add new records by scrolling down and typing a new record into a field.

Section 7. Summary

You have been introduced to two of the hottest trends in Web application development: Ruby on Rails and rich Internet application development using OpenLaszlo V3.2. Furthermore, you have learned how to create a rapid development environment using Eclipse to not only write and edit code but to automate setup and configuration of your Rails and Laszlo projects.

You have used the Rails and the OpenLaszlo plug-ins to sketch out a prototype of an application. In some cases, this might be sufficient for your needs. But in most cases, you're going to need to create more complex Rails models, edit your controllers, and write more complex Laszlo .lzx files. As you can see, however, using Eclipse can greatly reduce the amount of time associated with building rich Internet applications.

Resources

Learn

- Read "[Use Apache Derby in your OpenLaszlo applications, Part 1](#)" to learn how to use Apache Derby with OpenLaszlo.
- [Ruby-lang.org](#) is a great resource for anyone interested in Ruby.
- Be sure to stop by [RubyonRails.org](#) for further resources.
- For a developerWorks article about how Active Record and database wrappers work, read "[Crossing borders: Exploring Active Record](#)."
- Read "[Build rich Internet applications](#)" to learn about developing with Eclipse and OpenLaszlo.
- Read "[Develop SQL databases with Eclipse, SQLExplorer, and Clay](#)" to learn how to use Eclipse and the SQLExplorer plug-in to connect to any database that supports a JDBC driver.
- For more information about OpenLaszlo, visit [OpenLaszlo.org](#).
- For an introduction to the Ruby Development Tools plug-in for Eclipse, check out "[Using the Ruby Development Tools plug-in for Eclipse](#)."
- Check out the "[Recommended Eclipse reading list](#)."
- Browse all the [Eclipse content](#) on developerWorks.
- New to Eclipse? Read the developerWorks article "[Get started with Eclipse Platform](#)" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- Expand your Eclipse skills by checking out IBM developerWorks' [Eclipse project resources](#).
- To listen to interesting interviews and discussions for software developers, check out [developerWorks podcasts](#).
- For an introduction to the Eclipse platform, see "[Getting started with the Eclipse Platform](#)."
- Stay current with developerWorks' [Technical events and webcasts](#).
- Watch and learn about IBM and open source technologies and product functions with the no-cost [developerWorks On demand demos](#).
- Check out upcoming conferences, trade shows, webcasts, and other [Events](#) around the world that are of interest to IBM open source developers.
- Visit the developerWorks [Open source zone](#) for extensive how-to information,

tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Download the [OpenLaszlo Rails plug-in](#).
- Check out the latest [Eclipse technology downloads](#) at IBM [alphaWorks](#).
- Download [Eclipse Platform and other projects](#) from the Eclipse Foundation.
- Download [IBM product evaluation versions](#), and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.

Discuss

- The [Eclipse Platform newsgroups](#) should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The [Eclipse newsgroups](#) has many resources for people interested in using and extending Eclipse.
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

About the author

Robi Sen

Robi Sen is vice president of Department13 LLC, a boutique information and technology services company focusing on Fortune 1000 corporations and government organizations. He spends most of his time developing large enterprise-scale applications and has spoken extensively on the topics of enterprise application integration and service-oriented architectures.