FORM Tools

R1.0

ChiMu Corporation

1220 N. Fair Oaks Ave, #1314
Sunnyvale, CA 94089

Phone: 408 734-9068
Email: info@chimu.com

www.chimu.com

FORM Tools

Table of Contents

1 Overview

This document describes the FORM Tools. These include the FORM ExampleRunner, the FORM Test framework, and the FORM Preprocessor. FORM ExampleRunner is a GUI application for running the Learning FORM examples. FORM Test is a simple testing framework which can also be used to run the Learning FORM examples. The FORM Preprocessor is a tool to help you generate the required FORM binding information for your domain classes.

2 FORM ExampleRunner

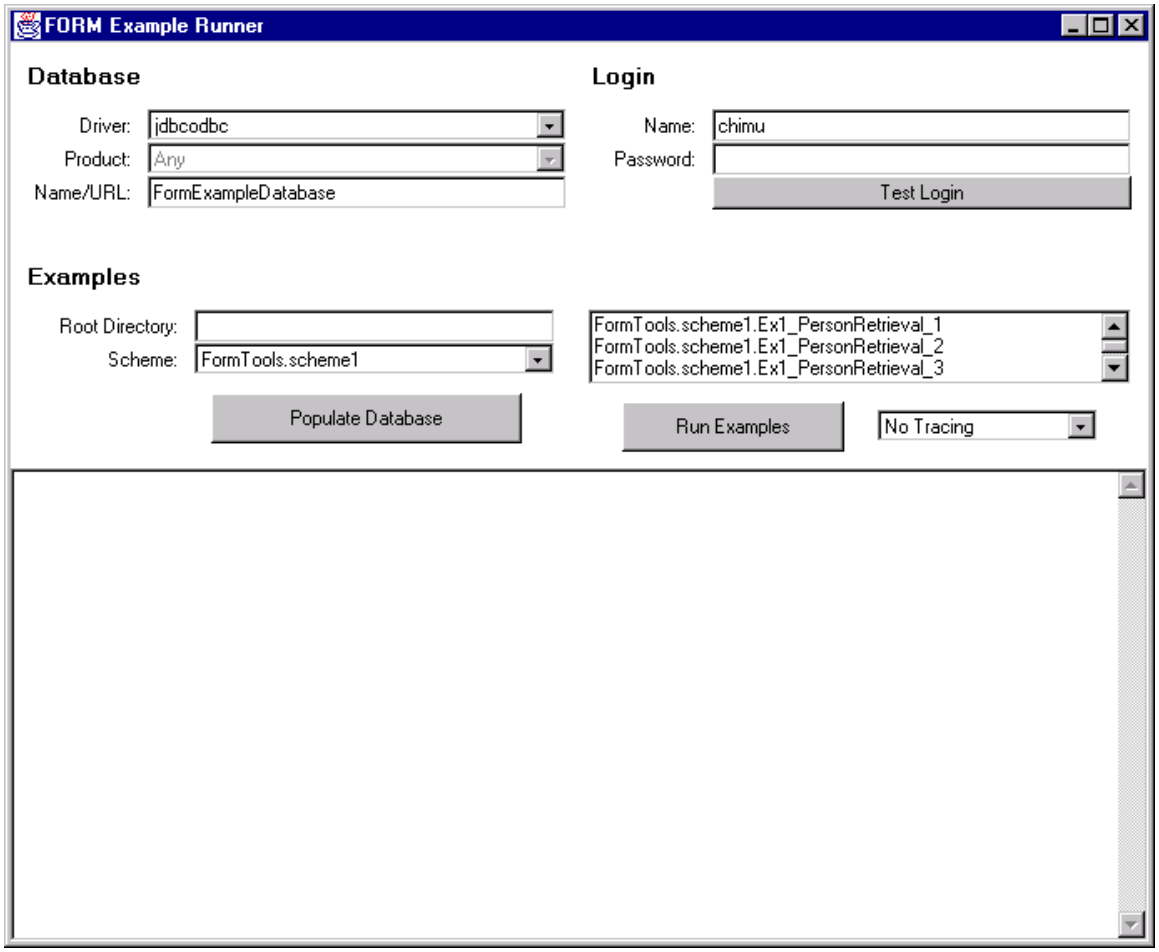
The first tool you will encounter is the FORM ExampleRunner. This provides a graphical user interface to run the provided FORM examples, or any examples you create[1].

To run the ExampleRunner, you can type into the command prompt:

```
> java com.chimu.formTools.examples.ExampleRunner
```

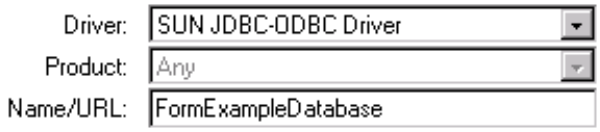
Alternatively, you can execute the batch file called 'FormExampleRunner.bat' in the 'examples' directory. Both will bring up the ExampleRunner application window.

The ExampleRunner application window is divided into four sections: Database and Driver information, User login information, Example population and selection, and the Results pane.

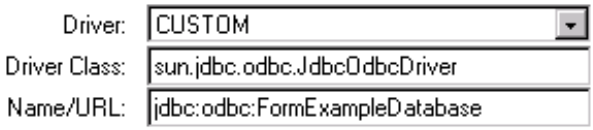


[1] See the chapter on the FORM Test framework for how to add new examples, or mimic the structure of the included examples.

2.1 Database and Driver Information

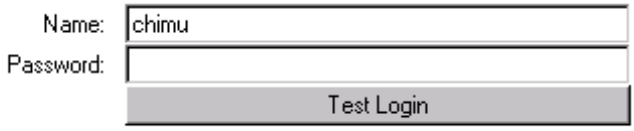


This section controls what database to log into and what driver to use. Depending on the driver chosen, the Database Product drop down box will activate and will allow you to choose a database supported by the particular driver. If a driver can work with any type of database (e.g. the ODBC driver), the product choice will be disabled. One driver choice is named 'CUSTOM' and allows you to use any driver by entering the fully qualified Class name of the driver.



A database can be identified either through the full JDBC URL appropriate for the particular driver, or, if you selected a FORM known driver (i.e. anything other than CUSTOM), FORM can build a full URL from just the database specific information. In the above example, FORM automatically prefixes the database name with 'jdbc:odbc:' to create a valid URL.

2.2 User Login

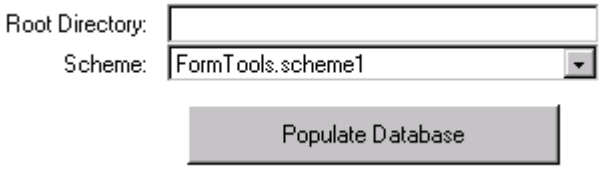


This section controls the database login information. After you have selected a database and entered your login information, you can test logging in to the database by clicking 'Test Login'. If the login is successful, the Result pane will show Catalog information about the database.

2.3 Example Population and Selection

This section allows you to populate the database and run examples. Examples are grouped by having an identical data model, which is referred to as a Scheme. Before you can use an example, you must select the scheme and populate the database with the appropriate tables and data for the scheme. After the database is populated you can select and run any of the examples that work with that scheme.

Selecting and Populating Schemes

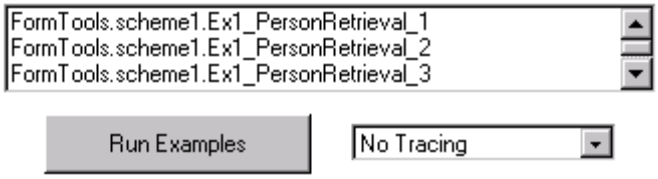


The 'Scheme' field displays the current selected scheme from the collection of available Schemes. If the 'Root Directory' field is empty, the 'Scheme' list will be populated from the com.chimu.formTools.examples package. Otherwise the Scheme list will be populated from the subdirectories under the RootDirectory. This allows you to create you own examples or modify the examples that come with the FORM distribution ('~chimu/examples/'). In order for FORM ExampleRunner to use the classes within the directories, the root directory must also be a part of the class path. Frequently the most convenient approach is to change to the desired root directory, launch the

ExampleRunner and then use '.' as the Root Directory name. This allows you to simply add '.' to your class path.

If you use one database for all the examples, you should click the 'Populate Database' button after selecting a scheme and before running any examples from a chosen scheme. By clicking on the 'Populate Database' button, the application will set up the database with the scheme's tables and data. This is accomplished by calling the 'DropExampleDatabase' and 'SetupExampleDatabase' classes in the scheme subdirectory. These two classes set up the database tables and populate the tables with data. If you create your own scheme directory, you will need to implement DropExampleDatabase and SetupExampleDatabase classes.

Selecting Examples



The examples list pane displays the possible selections of examples from within the scheme. It is a multi-selection list; you can select multiple examples and run them using the same established connection. Clicking on the 'Run Examples' button will execute all the examples selected in the list. The results of the examples will then be displayed in the Results pane.

Tracing level determines the details of the trace. The levels ranges from zero (no tracing) to three (most detailed tracing level). The trace is interlaced into the examples' output.

2.4 Results

Results pane displays the output of the example and any status information. If the tracing set, the output will also have tracing information interweaved through the result set.

If an example is run without tracing, the output may look something like this:

```
Test: ---- class com.chimu.formTools.examples.scheme4b.Ex4b_CompanyInsert_1 ----

Sennheiser with revenue: $ USD: $ 350000000.00000000

New Company Scarty Kitty, LTD with revenue: $ USD: $ 1000
```

Only the test name and the results of executing the example are displayed.

With tracing level set to 3, the output will look something like:

```
Test: ---- class com.chimu.formTools.examples.scheme4b.Ex4b_CompanyInsert_1 ----

  Test: **875837988873**

Driver Product   = JDBC-ODBC Bridge (SQLSRV32.DLL) [1.1001 (02.65.0201)]

Database Product = microsoft sql server [06.50.0201]

Recognized Database Product is: Microsoft SQL Server

FORM: Build ObjectInitializer From slots: [0:oid]=202 [1:name]=Scarty Kitty, LTD
[2:revenue]=1000.00000000 [3:employees]FORM: Build Objects from Rows: []=[] done.

Initialized: Scarty Kitty, LTD

FORM: Generate Identity

FORM: Build Row From slots: [0:oid]=204 [1:name]=Scarty Kitty, LTD
[2:revenue]=1000 done:[204 Scarty Kitty, LTD 1000 ]{revenue=2, id=0, name=1}
```

```
Scarty Kitty, LTD with revenue: $ USD: $ 1000.00000000

New Company Scarty Kitty, LTD with revenue: $ USD: $ 1000
```

The trace will show the resulting output as well as the SQL statements that were dispatched tot he database.

2.5 Other Features

The ExampleRunner will save your recently used preferences into a file called 'defaults.ser'. If you want to use a different file for saving preferences, pass it in as the first parameter to ExampleRunner:

```
> FormExampleRunner myDefaults.ser
```

2.6 Modifying and Creating Examples

You can create additional examples very easily within the 'examples' hierarchy.' Each scheme has its own 'ExampleAbsClass', which is the parent class of all examples within that scheme. In addition, all examples implement #run(jdbcConnection) method.

To create a new Example class, simply create a new class that inherits from the 'ExampleAbsClass' and implement the #run(jdbcConnection). In order for the example to be recognized by ExmpleRunner as a valid example, you must name your example class with a prefix 'Ex_' prefix, or 'Ex#_' where the '#' is the scheme number. Then the ExampleApplication will be able to recognize and include the new example in the available examples list pane.

2.7 Creating New Schemes

You can create a new scheme to be used with ExampleRunner. In order for ExampleRunner to recognize and access your scheme, each scheme you create should be in its own directory and should contain the following components:

- All classes within the scheme should be in the same package named after the scheme.
- A class named 'DropExampleDatabase'. This class should inherit from com.chimu.formTools.examples.DropExampleDatabase. Reimplement dropScheme() to contain appropriate drop table commands for your scheme.
- A class named 'SetupExampleDatabase'. This class should inherit from com.chimu.formTools.examples.FormExampleDatabasesAbsClass. Reimplement createScheme() and populateScheme() to create and populate data for your tables.
- A placeholder class named SchemeExamples.
- Domain classes for your scheme.
- Example classes should be named with 'Ex_' prefix or 'Ex#_' prefix where # is a number.
- The scheme directory should be located as a subdirectory of the root directory that you specify in the 'Root Directory' field of the ExampleRunner.

3 FORM Test

Included with the FORM Tools is a simple testing framework and a command line DatabaseTester that can log into the database and then execute multiple database test classes. FORM Test is important because most of the Learning FORM examples are written as DatabaseTests: in order to run the examples, you will need to use the DatabaseTester. FORM Test may also be useful for your own examples/tests that use JDBC. An even simpler test framework is included in the KernelTools package.

3.1 Introduction

FORM Test framework has two main components: a DatabaseTest, which contains an individual test that you want to run, and the DatabaseTester, which is a command line program that sets up the database and then runs one or more DatabaseTests. The Test framework is currently very simple; it is more of an Example framework. A test is given a JDBC connection and can then do whatever it wants. The DatabaseTester simply coordinates and error handles the individual tests. DatabaseTests can also be composed into suites of tests by producing DatabaseTests that use other DatabaseTests (which is very easy in Java).

3.2 Running Tests

To run a test you use the DatabaseTester. At its simplest, the DatabaseTester only needs the databaseName and one or more TestClass names.

```
> java com.chimu.formTools.test.DatabaseTester DatabaseName TestClassName
```

The DatabaseName should be appropriate to the given type of JDBC driver. The default driver is the JdbcOdbc driver from SUN, so the DatabaseName for the above example should be an ODBC source name. TestClassName is the full path name of a Java class. If the class is in the default package, the path name is just the class's name. Otherwise it will be a sequence (with periods as separators) of packages followed with the class name.

To find out the possible options for the DatabaseTester just execute it without any arguments

```
> java com.chimu.formTools.test.DatabaseTester
```

This will give you the complete listing of currently supported options, for example:

```
Usage:
    databaseTester [LoginControl] [DatabaseOptions] [TraceOptions] database
(TestClass)+
or
    java com.chimu.formTools.test.DatabaseTester ...

Login Control:     [-l loginName [-p password]]
    -l <loginName>
    -p <password>

Database Options:  [-dr driverProduct [-da databaseProduct]]
    -dr <driver>      The identifier for the jdbc driver. One of:
                        jdbcodbc[default], fastforward, webconnect, jconnect
    -da <database>    The identifier for the database product. One of:
                        oracle, sybase, mssqlserver, db2, or access

Trace Options:     (-t | -tl traceLevel) [-q]
    -t                Enable Tracing at detail level 1
    -tl <level>       Enable Tracing at detail level <level>
    -q                Be quiet.  Do not include test name trace in output.

Arguments:
    database          The database identifier appropriate to the driver product
    TestClass         The test class name or full path name
```

Running the Connection Info Test

If you do have ODBC on your machine and have setup the FormDm1 name to point to the FORM example domain model, then the following command:

```
> java com.chimu.formTools.test.DatabaseTester FormDm1
                  com.chimu.formTools.test.ConnectionInfoTest
```

This should return information about your database. For example:

```
Test: ---- com.chimu.formTools.test.ConnectionInfoTest ----
Driver Product   = JDBC-ODBC Bridge (SQLSRV32.DLL) [1.1001 (02.65.0213)]
Database Product = Microsoft SQL Server [06.50.0201]

Catalogs (database)
FormExampleScheme1a

Schemas (owner)
dbo
```

If during installation, you set up the batch or shell files that came with FORM, you can simplify the above command line to

```
> DatabaseTester FormDm1 com.chimu.formTools.test.ConnectionInfoTest
```

We will assume this simplification for the rest of the chapter, but if the batch files are not available you can replace the short form with the longer, universal form.

Different Drivers and Databases

A problem with the example above may be that you do not have an ODBC driver. You can tell the DatabaseTester to use a different driver by setting the "-dr" option. Check the command line information for the current options, but drivers might include 'fastforward', 'webconnect', and so on. Most of these drivers will also require knowing what type of database you are using, which can be set using the "-da" option. Each driver has different types of formats that it expects for the database name, which will usually be the part of the URL to the right of the second (and final) colon. Some drivers use an Internet based database name '//1.2.3.4:port/databaseName' and other drivers automatically know the database an simply need a database name.

Universal driver and database specification

If the DatabaseTester does not know about your specific driver you can instead tell it a driver class and a JDBC URL. You specify the driver class using the '-drc' option. The database name should be a fully formed JDBC URL. For our above example we would have:

```
> DatabaseTester –drc sun.jdbc.odbc.JdbcOdbcDriver jdbc:odbc:FormDm1
                  com.chimu.formTools.test.ConnectionInfoTest
```

Login Name and Password

You can specify the login name and password through the '-l' and '-p' options respectively. If you do not specify a login name or password the DatabaseTester will attempt to login to the database with the standard product-specific demo name/password pair (e.g., 'sa' for Sybase).

Tracing

You can enable tracing for tests with the '-t' and '-tl level' options. The main DatabaseTester will put profiling information in and around each test if tracing is turned on. Each test may or may not respond to tracing levels. See the DatabaseTest Framework for more information on tracing information given to a test.

3.3 Creating Tests

Creating a DatabaseTest is simply creating a class that implements the DatabaseTest interface.

```
public interface DatabaseTest {
    public void run(Connection jdbcConnection);
    public void setupTracing(PrintStream newTraceStream, int
           newTraceLevel);
}
```

Any class that implements this interface can be used as a DatabaseTest with the DatabaseTester. To make creating tests a little easier (and support future extensions of DatabaseTest), FORM Test provides DatabaseTestAbsClass. This abstract class implements the setupTracing method and remembers the traceStream and traceLevel, leaving only the run method up to each specific test class. FORM Test also provides a FORM specific abstract class FormDatabaseTestAbsClass, which has FORM specific code that helps with writing FORM tests.

"Run" Method

The #run method is the main entry point for the test classes. All test classes should implement this method to do what the test requires. The interface definition is:

```
public void run (Connection jdbcConnection);
```

A Simple Test

For example, we can create a simple HelloWorld test. We will extend FormDatabseTestAbsClass to get the tracing implementation and we implement the #run method to print a greeting on 'System.out'.

```
import java.sql.Connection;

import com.chimu.formTools.example.*;
import com.chimu.formTools.test.*;

public class HelloWorld extends DatabaseTestAbsClass {
    public void run(Connection jdbcConnection) {
        System.out.println ("Hello World");
        tracePrintln("We printed the Hello World message");
    }
}
```

If we execute this test with no options:

```
> DatabaseTester FormDm1 HelloWorld
```

We would get:

```
Test: ---- HelloWorld ----
Hello World
```

If we turned on tracing:

```
> DatabaseTester –t FormDm1 HelloWorld
```

We would get:

```
Test: ---- HelloWorld ----
Hello World
trace: We printed the Hello World message
```

FormDatabaseTestAbsClass

FormDatabaseTestAbsClass extends from DatabaseTestAbsClass and also has FORM specific code that helps with writing FORM tests. Some of the helper methods are listed below for your reference; a complete listing can be found in the ChiMu Tools API documentation.

createOrm(Connection aConnection)	Creates and remembers an Orm (stored in 'orm') and a DatabaseConnection (stored in 'dbConnection').
newObjectMapper(Table aTable)	
newDistinguishingObjectMapper(Table aTable)	
newAssociationMapper(Table aTable)	
newTable(String tableName)	

FORM Specific Sample Tests

See the examples in 'Learning FORM' for some FORM specific sample tests.

4 FORM Preprocessor

The FORM preprocessor makes managing the code for domain object to FORM binding easier. The FORM preprocessor extracts information from your source file and regenerates the source file with extra binding code within it. If you don't like the results, you can edit them after the generation; disable (or not enable) that portion of the generated code; or you can choose not to use the preprocessor at all and use your own approach to put in the needed binding code.

Currently, the FORM preprocessor annotates a source file in two main areas: adding 'unstub()' methods onto the public methods of the object and generating the FORM to object binding information methods.

4.1 Unstubbing

Before a replicated object can be accessed, you must make sure that the object has been fully retrieved from the database and not just as a stub object. This is accomplished by checking the state of the current object to see if it is a stub object, and , if so, asking the object's mapper to unstub (fully initialize) the object. The only FORM required method to call is:

```
public void unstub(Object object);
```

The rest is flexible, though the recommended approach is to have the domain object itself be responsible for making sure it is a full replicate when needed. This can be accomplished by having a method unstub() in the domain object (usually in the abstract superclass) that looks like:

```
unstub() {
    if (this.isStub()) this.mapper().unstub(this);
}
```

where the 'isStub()' check is a very fast state check (either a flag test or instance variable test).

If the object itself is responsible for making sure that it is not a stub object, we recommend that the domain object also hide the knowledge of having to unstub from other objects. Since the domain objects must be completely instantiated (i.e., unstubbed) whenever another object calls, the recommended place to put these 'unstub' methods are the public or package methods. The public or package methods are good places; since all other objects, with the exception of objects of the same class, must call through these public or package methods.

The FORM preprocessor adds all the necessary unstub's to your class's methods. The FORM preprocessor will add an 'unstub()' method to any (non-static) public or package method that does not:

1. have an unstub() as the first statement in the method already, or
2. have '//<DontAutoUnstub>' as one of the very first comments in the method

4.2 Binding Information

The FORM preprocessor can generate the binding to the domain object. It does not generate any information regarding the database or the mapping between the object and the database. This is left for the application. The FORM preprocessor generates binding information by reading an object's field declarations. From the non-transient fields, the preprocessor generates:

1. Object initialization and extraction methods, and
2. Static functors for constructing the object and retrieving information from the object.

These are the main pieces needed for specifying the database binding itself.

The application developer decides the location of the generated section. The FORM preprocessor looks for a section tag starting with '//<GenerateObjectBinding{' and ending with '//}GenerateObjectBinding>' to place the generated binding information. This section would normally be blank before the first generation:

```
//<GenerateObjectBinding{
//}GenerateObjectBinding>
```

After each generation the empty section between the tags will be replaced by the generated methods and comment headers. This generated section looks something like the following:

```
//<GenerateObjectBinding{
    //FORM Preprocessor Version 1.4 ran on: Tue Jul 08 20:26:11 PDT
           1997

    //*********************************************************
    //(P)************ Initializing & Extracting ***************
    //*********************************************************

    /*friend:FORM*/ public void
           form_initState(com.chimu.kernel.collections.KeyedArray
           slotValues) {
        super.form_initState(slotValues);

        this.name = (String) slotValues.atKey("name");
        this.email = (String) slotValues.atKey("email");
        try{ this.height = ((Integer)
           slotValues.atKey("height")).intValue();  } catch (Exception
           e) {this.height=0;};
    }

    /*friend:FORM*/ public void
           form_extractStateInto(com.chimu.kernel.collections.KeyedArr
           ay slotValues) {
        super.form_extractStateInto(slotValues);

        slotValues.atKey_put("name",this.name);
        slotValues.atKey_put("email",this.email);
        slotValues.atKey_put("height",new Integer(this.height));
    }

    //*********************************************************
    //(P)******************* Constructor **********************
    //*********************************************************

    protected PersonClass(com.chimu.form.mapping.CreationInfo cInfo) {
        super(cInfo);
    }

    //*********************************************************
    //(P)******************* Class Info ***********************
    //*********************************************************

    static Class myClass() {
        return PersonClass.class;
    }

    static protected com.chimu.form.mapping.CreationFunction
           form_creationFunction() {
        return new com.chimu.form.mapping.CreationFunction() {
            public com.chimu.form.mapping.MappedObject
           valueWith(com.chimu.form.mapping.CreationInfo cInfo){
                return new PersonClass(cInfo);
            }
        };
    }

//}GenerateObjectBinding>
```

Note that the generation section **must be** after all the instance variable declarations. It is generally convenient to place it near the instance variables themselves or at the end of the file.

5 Where to go next

This document gave a very brief overview of FORM architecture, capabilities, and concepts. This overview should help you to understand FORM and help guide you to what you want to learn in more detail. Each of the topics discussed here is given a much more thorough description in *Learning FORM*. If you are more interested in examples, *Learning FORM* and the source code to the examples provide more detail. Finally, if you would like a higher level picture of object-relational mapping and architecture, you could look at *A Good Architecture for Object-Oriented Information Systems*, *Foundations for Object-Relational Mapping*, and the references contained in these two documents.

- *Getting Started with FORM* describes how to install FORM.
- *Learning FORM* is the main detailed documentation of FORM.
- The Examples show code that works with FORM.
- The *FORM API* Javadoc gives a complete description of the FORM interface.
- The *FORM Tools* manual describes the command line and graphical FORM tools in detail.

5.1 ChiMu Corporation References

Architecture	*A Good Architecture for Object-Relational Mapping*
FORM Tools	*FORM Tools*
Foundations	*Foundations of Object-Relational Mapping*
Introduction	*Introduction to FORM*
Kernel	*Kernel Frameworks*
Learning	*Learning FORM*
Standards	*Java Development Standards*
Starting	*Getting started with FORM*

ChiMu Corporation

1220 N. Fair Oaks Ave, #1314
Sunnyvale, CA 94089

Phone: 408 734-9068
Email: info@chimu.com

www.chimu.com