

# 7

## *Connecting to the Database*

When working with a data source, or at the database or adaptor levels, you must establish a connection to the database before you can interact with it. This chapter shows you how to make that connection in three different ways: When you're working with a data source, when you're working primarily at the database level, and when you're working primarily at the adaptor level. When you're working at the database or adaptor levels, you need to load a model file before you can connect to the database. Therefore, this chapter also shows you how to locate and load a model file into memory.

If you're working with a controller, the model is loaded and the database connections are established for you by the Framework; thus, you can skip this chapter.

This chapter is organized into the following sections:

- “Loading a Model” discusses how you go about loading a model into memory.
- “Connecting Using a Data Source” shows you how to connect to the database using a data source.
- “Connecting at the Database Level” shows you how to connect to the database using database-level objects.
- “Connecting at the Adaptor Level” shows you how to connect to the database using adaptor-level objects.
- “The Connection Dictionary” talks about the specific items that your application needs to log in to the database server, and how you can obtain that information from the user.

This chapter only discusses how you establish a connection to the database. Operations subsequent to establishing that connection—such as fetching or deleting objects—are covered in later chapters.

### **Loading a Model**

If you're working at the adaptor level or at the database level you need to load a model before you can establish a connection to the database. A model file stores a mapping between the database schema and your enterprise objects, along with information needed to log into the database server. You typically create model files with EOModeler, although you can construct them programmatically.

Model files must be stored in one of the directories that make up the standard resource directory search path. This path consists of the following directories (in this order):

- The application's main bundle
- ~/Library/Models
- /LocalLibrary/Models

Given the base name of a model file, EOModel's **findPathForModelNamed:** class method searches the directories in this path and returns the full file name of the requested file. You can then supply this name to **initWithContentsOfFile:** to initialize a newly-allocated model object.

The following code excerpt shows how to instantiate and initialize a model object with the contents of a model file named **People.eomodel**:

```
NSString *modelPath;
EOModel *peopleModel;

modelPath = [EOModel findPathForModelNamed:@"People"];
peopleModel = [[EOModel alloc] initWithContentsOfFile:modelPath];
```

For instructions on using EOModeler to create a model file, see Chapter 5, “Using EOModeler.” Chapter 11, “Exploring and Constructing Models,” shows you how to examine and create models. Finally, for information on model objects and the methods you use to manipulate them, see the EOModel class specification in the *Enterprise Objects Framework Reference*.

## Connecting Using a Data Source

If your application uses enterprise objects and the default updating, snapshotting, and uniquing behaviors, the easiest way to programmatically connect to a database is by creating a data source. To create a database data source, you simply instantiate an EODatabaseDataSource object and initialize it with the base name of your model and the name of your entity.

The following code excerpt shows one way to connect to the database, given a model file named **People.eomodel** that contains an **Employee** entity:

```
EODatabaseDataSource *myDataSource;

myDataSource = [[EODatabaseDataSource alloc]
    initWithModelName:@"People" entityName:@"Employee"];
```

Note that you only supply the base name of the model file (the file name without extensions) to **initWithModelName:entityName:**. The model file must be located in one of the directories that make up the standard resource directory search path (see “Loading a Model” for more information).

In the above example, the data source was initialized with the **initWithModelName:entityName:** method. To set up a second data source object using the same channel (for a different entity in the same database), you can either use the **initWithDatabaseChannel:entityNamed:** method and supply the database channel object that was created when the first channel was opened, or you can use **initWithModelName:entityName:** and supply the same model name. The latter works because **initWithModelName:entityName:** first checks to see if the model is already loaded; if it is, the database data source is initialized with the loaded model and pre-existing channel.

Note that you can only have a single fetch or update operation in progress on a single channel. If you need to perform multiple operations at one time, create other data sources that use separate channels.

For more information on the methods you use to initialize an EODatabaseDataSource object, see the EODatabaseDataSource class specification in the *Enterprise Objects Framework Reference*.

## Connecting at the Database Level

If your application is going to fetch enterprise objects using a fetch loop, you need to connect to the database at the database level (see Chapter 8, “Retrieving Records” for more information on fetch loops). You make this connection by opening a channel to the database, using an `EODatabaseChannel` object to represent the channel. See the `EODatabaseChannel` class specification in the *Enterprise Objects Framework Reference* for a complete description of database channel objects and methods.

To open a channel with a database channel object, follow these basic steps:

1. If you don't have a model already loaded, load or create one (see “Loading a Model”).
2. Instantiate a database object and initialize it from your model.
3. Instantiate a database context object and initialize it from your database object.
4. Instantiate a database channel object and initialize it from your database context object. Send **autorelease** to it if you intend to vend it and not keep your own reference.
5. Verify that the adaptor associated with your database object has a valid connection dictionary (see “Verifying the Connection Information”).
6. Open the channel.

The following code excerpt illustrates the above steps, given a model that's already been loaded:

```
EOModel *myModel;      /* Assume this exists */
EODatabase *myDatabase;
EODatabaseContext *myContext;
EODatabaseChannel *myChannel;

myDatabase = [[EODatabase alloc] initWithModel:myModel];
myContext = [[EODatabaseContext alloc] initWithDatabase:myDatabase];
myChannel = [[[EODatabaseChannel alloc]
              initWithDatabaseContext:myContext] autorelease];

/* Verify connection information here. */

if ([myChannel openChannel] == NO) {
    /* Handle error... */
}
```

## Connecting at the Adaptor Level

You connect at the adaptor level primarily if you're writing an application, such as a report writer, that is concerned only with raw data, and not with the methods that an enterprise object couples to that data. At the adaptor level, each record is returned as a dictionary.

You connect at the adaptor level by opening a channel to the database, using an `EOAdaptorChannel` object to represent the channel. See the `EOAdaptorChannel` class specification in the *Enterprise Objects Framework Reference* for a complete description of adaptor channel objects and methods.

To open a channel with an adaptor channel object, follow these basic steps:

1. If you don't have a model already loaded, load or create one (see “Loading a Model”).
2. Create an adaptor object from your model.
3. Create an adaptor context object from your adaptor object.
4. Create an adaptor channel object from your adaptor context object and retain it (unless you're going to vend it). This keeps the adaptor and adaptor context objects from being deallocated until

you release the adaptor channel object.

5. Verify that the adaptor has a valid connection dictionary (see <sup>a</sup>Verifying the Connection Information<sup>o</sup>).
6. Open the channel.

The following code excerpt illustrates the above steps, given a model that's already been loaded:

```
EOModel *myModel;      /* Assume this exists. */
EOAdaptor *myAdaptor;
EOAdaptorContext *myContext;
EOAdaptorChannel *myChannel;

myAdaptor = [EOAdaptor adaptorWithModel:myModel];
myContext = [myAdaptor createAdaptorContext];
myChannel = [[myContext createAdaptorChannel] retain];

/* Verify connection information here. */

if ([myChannel openChannel] == NO) {
    /* Handle error... */
}
```

## The Connection Dictionary

An adaptor object maintains a dictionary that contains connection information used when connecting to the database server. The keys of this dictionary identify the information the server expects; the values associated with those keys are the values that the adaptor tries when logging into the database.

The dictionary keys required by the Oracle, Oracle7, and Sybase adaptors are:

Oracle Adaptors	Sybase Adaptor
hostMachine	hostName
serverID	databaseName
userName	userName
password	password

When you initialize an adaptor from a model, any connection information stored with the model is copied into the adaptor object.

## Verifying the Connection Information

Before opening the channel, you should verify that you have sufficient connection information to log into the database server. Sending a **hasValidConnectionDictionary** message to the adaptor verifies that the adaptor has the proper information needed to log into the server. If the connection dictionary doesn't have valid information (for example, it's common to leave the user name and password unspecified in the model file), you can invoke **runLoginPanelAndValidateConnectionDictionary** to display a login panel that allows the user to enter the proper connection information. The following code illustrates how to do this:

```
myAdaptor = [myDatabase adaptor];
if ([myAdaptor hasValidConnectionDictionary] == NO &&
    [myAdaptor runLoginPanelAndValidateConnectionDictionary] == NO) {
    /* Don't log in. */
}
```

The methods you use to verify connection information are only supplied at the adaptor level. To

obtain the corresponding adaptor object for a database object, use EODatabase's **adaptor** method. From a data source object, you can obtain the adaptor object it uses with:

```
theAdaptor = [[[myDataSourceObject databaseChannel] adaptorChannel]
               adaptor];
```