

Multiuser Chapter 2

Protecting Your Server

Quote: "What is your name?"

" I am Arthur, King of the Britons"

"What is your quest?"

"I seek the Holy Grail"

"What is the windspeed velocity of the swallow?"

Monty Python's "The Holy Grail"

Running a server on the Internet can be a risky business. Your aim is to provide a service to bona fide users. Unfortunately, a tiny minority of internauts may find other ways of exploiting your server, ranging from the practical joke to the criminal. As far as the Multiuser Server is concerned, you want to prevent unauthorized movies using your bandwidth, server resources and disk space. With the default settings, all the hacker needs is the IP address of your server and the port number the MultiuserServer is running on. While it is impossible to protect your server completely and still provide a service, taking a number of basic precautions will make your server a less appealing target than the next one. In this chapter, you will find out how to create a simple database of trusted users and how to limit unwanted access to your server without limiting the service you provide.

Note: In the text that follows, I refer to hackers as "he". Either the she-hackers are much more careful not to get caught than the males, or there are far fewer of them.

(1)Gaining Access to the Server

Neither friend nor foe can connect to your server unless they know its vital statistics. The Multiuser Server is an application that runs on a physical machine. Think of the machine as a tower block in the business part of town, and the Multiuser Server application as a company with one or more offices in the tower block. To connect to the server, the user must know both the

address of the physical machine (its IP address), and the number of one of the offices that the server application works from (its port number). If the server is properly protected, the user must also show an identity card at the door and may be refused entry if the card is unknown or suspicious.

(2)IP Addresses

Each physical machine connected to a network (Internet or intranet) has an IP address. "IP" stands for Internet Protocol. An IP address is a set of four numbers separated by dots, such as 123.45.67.89. Each of the numbers will always be in the range 0-255. A computer that is not connected to a network may have an IP address of 127.0.0.1 or 0.0.0.0 by default.

There are several ways to find out the IP address of your computer. (Your IP address is likely to be different from the values shown below).

- Launch the MultiuserServer application on your machine, then select the menu State > Server. The IP address will be displayed in the server's console, as shown in Figure 1.

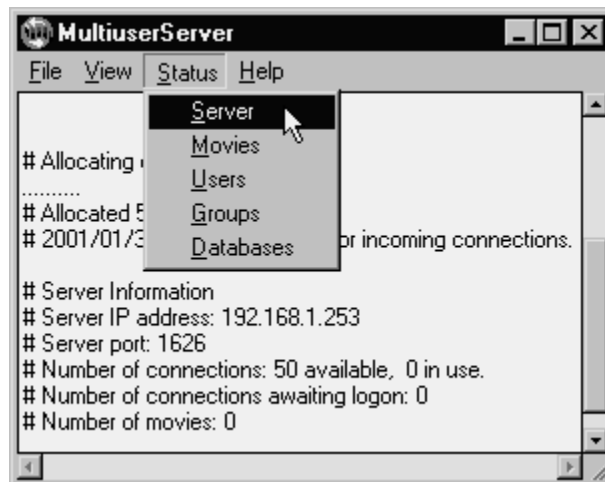


Figure 1: Determining the MultiuserServer's IP address and port number

- Type the following command into the Director's Message window:

```
put xtra("Multiuser").new().getNetAddressCookie(FALSE)  
-- "127.0.0.1"
```

- On Macintosh, choose the menu Apple > Control Panels > TCP/IP. The TCP/IP window should open. If you are permanently connected to a network, the IP address should be displayed. If you use a dial-up connection to connect to the Internet, you may find that your IP address is determined by the server you connect to, in which case it may be different every time you connect.
- On Windows, choose Start > Run then type "winipcfg" (without the quote marks). The IP Configuration window should open. If your IP address is not displayed directly, try a different configuration in the dropdown list. As on Macintosh, your machine may not have a fixed IP address if you are not connected to a network on a permanent basis.

The form of your machine's IP address depends on the type of network your machine is connected to. If it is connected to the Internet, the IP address needs to be unique. All machines on the same local network must have distinct IP addresses. However, two machines on two separate local networks can have the same local IP address, so long as they do not have a direct connection to the Internet. One range of IP addresses is reserved for local networks, another for Internet use.

(2)Ports and Protocols

Certain applications, such as the MultiuserServer, open ports when they run. A port is a door through which a user on another computer can exchange data with the computer running the application. Only one application at a time can use a particular port. Unless it is providing a basic service such as http or ftp access, the application which opens the port is likely to require a username and password before it will allow any data to enter or leave.

Even once a connection is established via the port, the application that opened the port will only know how to deal with certain types of data. For instance, the Multiuser server can only handle incoming commands in the format sent by the Multiuser xtra, and it will only return a property list in reply. If incoming data does not conform to the expected protocol, the application should ignore it.

(2)Firewalls

A firewall is a software application designed to limit access to the open ports. Without a firewall, anyone can send data to an open port, and try to retrieve information in return. There is nothing to prevent a hacker from testing a whole series of usernames and passwords in order to gain admittance. If you install a firewall, you can ensure that only users at registered IP addresses or domains can connect. The hacker thus doesn't get a chance to run his tests... unless he is working from a trusted domain.

Each incoming packet of data is accompanied by information on who sent it. The firewall reads this information, and rejects any packets of data with the wrong credentials. (It may also take stronger dissuasive action). You can configure the firewall to react differently for each port. For instance, you are likely to leave port 80 open to all-comers for http access, and perhaps also port 21, which handles the ftp protocol. If you are creating a multiuser application for public use, you would also want to allow anyone to connect to port 1626 (or whichever other ports you have chosen).

You'll find below a simple server-side script that acts as a firewall for your Multiuser Server. You will be able to customize this for your own needs.

(2)Proxies

A proxy is a server application. It can have two functions. It can store frequently used data in a cache so that you don't have to search the Internet for it each time you need it. This is the better-known function, but this doesn't really concern us here.

A proxy can also allow you to connect to a network when your IP address is in the range allocated to local networks: 192.168.xxx.xxx. An internet-based server cannot respond to such an address since it is not unique: another user in another local network can have the same number. Most companies have a local network. If your users work for such a company, they will not have direct access to the Internet: all their interactions must pass through the proxy server.

At the risk of oversimplifying, the proxy server will define which ports the user can connect to the Internet on, and which ports will be visible from outside. Certain public Internet Service Providers, such as AOL, also use a proxy server system.

(1)Registered Users Only

The simplest way to protect your server is to create a database of trusted users. Each user will have a log-on name and a password. With such a system in place, the only way an unauthorized user could gain access to the server would be to discover a logon/password pair. The protection could be strengthened by requiring the trusted user to log on from a trusted IP address.

The question is: how should the log-on name and password be attributed? The safest system is not necessarily the most user-friendly, and our purpose here is to add protection discretely. The system I shall show you in this chapter allows users to set their own log-on name and password the first time they connect. This system has its Achilles heel. In order to create his or her own database record, the new user will momentarily have to be granted administrative privileges. We will do our best to protect this Achilles heel.

This users' database can be used as more than just a security system. You will also be able to store information concerning your users, such as their subscription renewal date or an activity log. As you will see in later chapters, more complex database systems rely on the same techniques.

In the text that follows, I don't make the distinction between the server and its xtras. Technically, the multiuser features that you will be using to create the database are controlled by the DBObjects Xtra. The database itself is stored in the files in the DBObjectFiles folder.

(2)A Test Doomed to Failure

Creating a user database is both simple and tortuous. Simple, because the Multiuser Server is designed to handle database records. Tortuous because the information on how to set about creating the database is scattered in a variety of places.

Let's start with a test. The official PDF documentation, "Using Shockwave Multiuser Server" gives the syntax for creating a user record in the `createUser` entry in the "Server, group and database commands" section of chapter 3. Figure 2 shows the suggested example.

```
Example This statement creates a new DBUser object for the user Bob with the password MySecret and a user level of 40:  
  
errCode = gMultiuserInstance.sendNetMessage("system.DBAdmin.createUser", "anySubject",  
[#userID: 'Bob', #password: 'MySecret', #userlevel: 40])
```

Figure 2: The official example for the system.DBAdmin.createUser command

However, if you test this, the result is not what you might expect. Perhaps the easiest way to test is to open the movie `MUSyntax.dir`, which you will find in the folder `Multiuser\DSWMedia\` on the CD-Rom that accompanies this book.

Launch the MultiuserServer application. If necessary, use the State > Server menu to check the IP address and port number that the server is running on. Now launch the MUSyntax movie, and ensure that `connectToNetServer` is selected in the commands dropdown list. Modify the connection parameters if necessary, and then click on the "Execute" button.

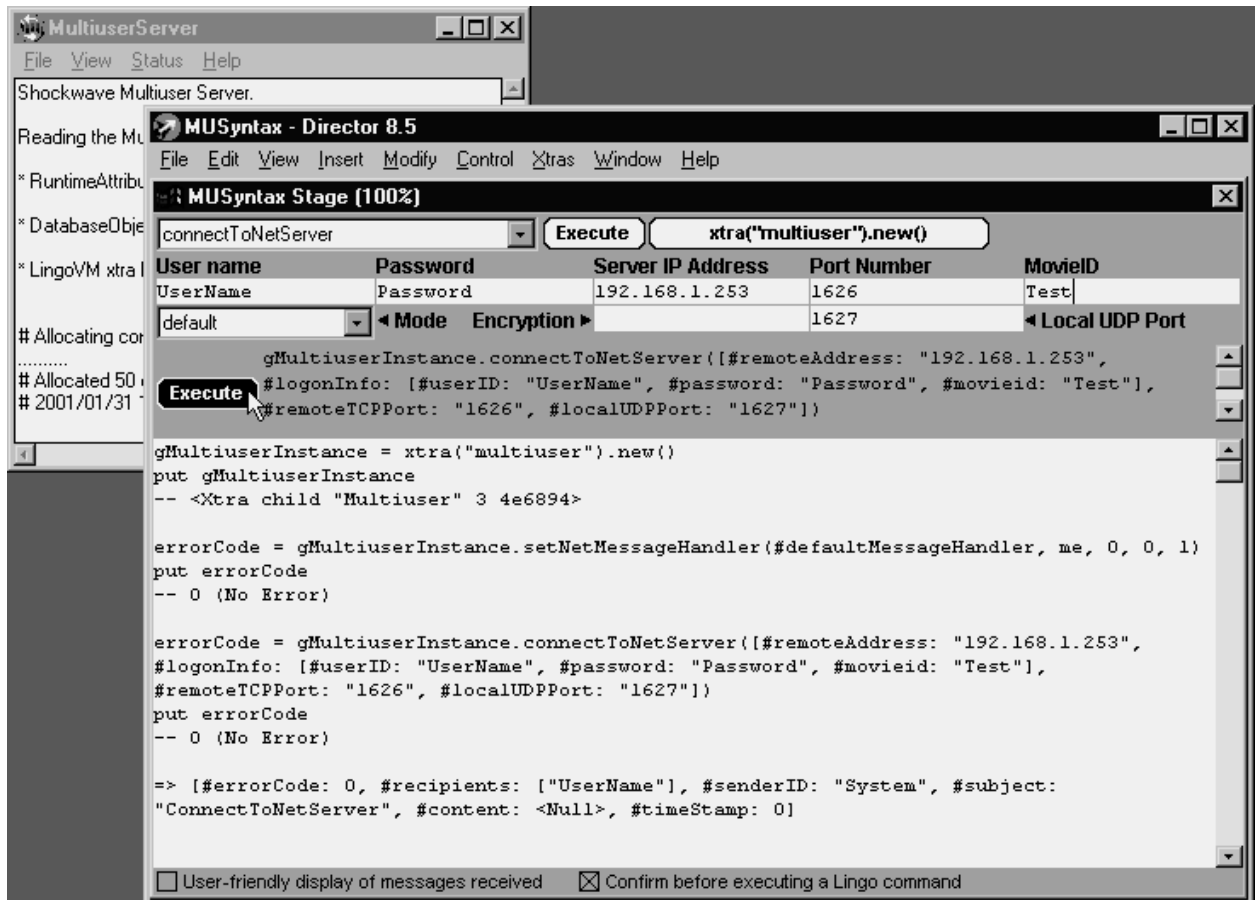


Illustration 1

In the console, lines beginning with `=>` indicate incoming messages sent by the Multiuser Server. If the value of the `#errorCode` property is anything other than zero, a problem occurred. If this happens when you try to connect to the server, check that the connection parameters are valid.

Once you have connected successfully to the server, choose `sendNetMessage` in the Commands dropdown list, and `system.DBAdmin.createUser` in the list of recipients recognized by the server. You can accept the default parameter values or modify these to

correspond exactly to those given in the example in Figure 2. In either case, the result will be the same. When you click on the "Execute" button, the console should appear as in Figure 3:

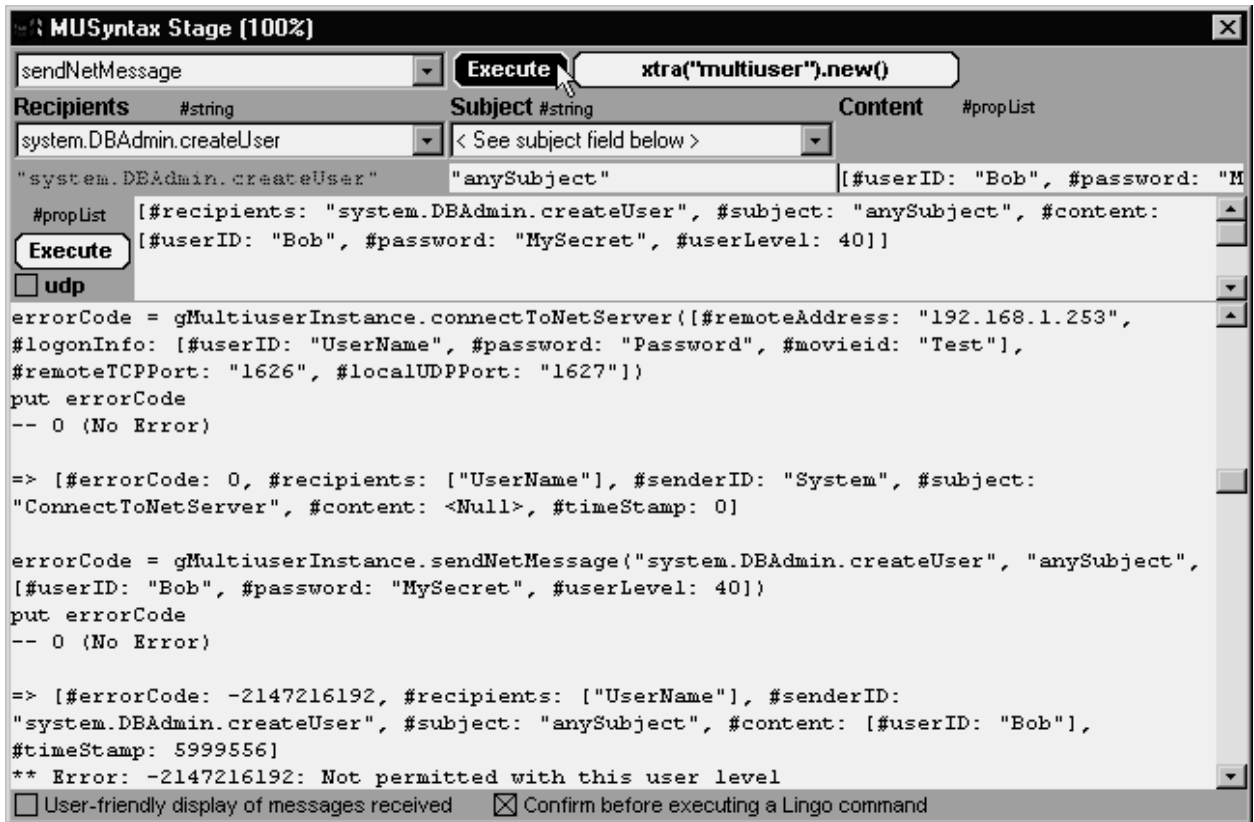


Figure 3: The system.DBAdmin.createUser command

What's this? "Not permitted with this user level"? What is a user level, what level do you need to execute the `createUser` command, and how do you acquire such a level?

(2)Multiuser.cfg

The first thing that the MultiuserServer does when it is launched is to read the Multiuser.cfg file, which lives in the same folder. Let's do the same. You can open Multiuser.cfg in any word processor. You'll see that most of the lines in the file begin with the pound character (#). This acts as a comment character: the MultiuserServer will only execute lines that do not start with this character.

Scroll down to line 70 or so, where you will find the following text.

```
#=====
# Default user level for someone logging on to the server.
# This is used if you do not specifically authorize logons
# with a database.
#=====
# DefaultUserLevel = 20
```

This last line is commented out. However, since no DefaultUserLevel is defined, the MultiuserServer will implicitly assign a value of 20, as a fallback default. You can already deduce that 20 is not considered a high enough value for the `createUser` command, but you'll need to look a little further to determine what the user level value should be if you are to create a record for your new user.

Look at line 280 and the following lines:

```
#=====
# Set the userlevels required to execute various commands
#=====
# Admin commands
XtraCommand = "System.DBAdmin.CreateUser 80"
XtraCommand = "System.DBAdmin.DeleteUser 80"
XtraCommand = "System.DBAdmin.CreateApplication 80"
XtraCommand = "System.DBAdmin.DeleteApplication 80"
XtraCommand = "System.DBAdmin.CreateApplicationData 20"
```

```
XtraCommand = "System.DBAdmin.DeleteApplicationData 20"  
XtraCommand = "System.DBAdmin.GetUserCount 80"  
XtraCommand = "System.DBAdmin.GetUserNames 80"
```

Though this is not very explicit, you should be able to work out that the default user level of 20 allows you to use commands such as `System.DBAdmin.CreateApplicationData` and `System.DBAdmin.DeleteApplicationData`. However, other commands, and notably `System.DBAdmin.CreateUser` which concerns us here, require a user level of 80.

This explains why the server rejected your request just now, with the excuse that the `createUser` command is Not permitted with this user level.

(2)Obtaining privileges

So how do you provide the current user with a privileged user level? There are several solutions. Let's start with the worst one.

First, you could set the `DefaultUserLevel` parameter to 80 in line 76 of the `Multiuser.cfg` file. Instead of...

```
# DefaultUserLevel = 20  
... you could use...
```

```
DefaultUserLevel = 80
```

Try it. Make the change shown above and save the `Multiuser.cfg` file. Now restart the server, and reconnect to it via the `MUSyntax.dir` movie. Choose `sendNetMessage` in the Commands dropdown list, and `system.DBAdmin.CreateUser` in the Recipients dropdown list, just as you did earlier. Then click on the "Execute" button. This time the server is happy to do your bidding. Figure 4 shows your request and the server's reply to it:

```
errorCode = gMultiuserInstance.sendNetMessage("system.DBAdmin.createUser", "anySubject",  
[#userID: "Bob", #password: "MySecret", #userLevel: 40])  
put errorCode  
-- 0 (No Error)  
  
=> [#errorCode: 0, #recipients: ["User name"], #senderID: "system.DBAdmin.createUser",  
#subject: "anySubject", #content: [#userID: "Bob"], #timeStamp: 6464711, #udp: 0]
```

Figure 4: Creating a new user record

Success! But what is the cost?

Caution: The user level that you set in the Multiuser.cfg file will be used for all movies that connect to the server. That means both the movies you yourself create and those written by hackers who may try to exploit your server. The modification that you have just made gives any such hacker extensive privileges. This is likely to be something you would prefer to avoid. Before going any further, you should set the DefaultUserLevel parameter in line 76 of the Multiuser.cfg file back to its default value of 20. Relaunch the server so that it uses the corrected configuration.

A slightly better solution could be to lower the userlevel required to create a new user record. You could modify line 284 of the Multiuser.cfg from...

```
XtraCommand = "System.DBAdmin.CreateUser 80"
```

... to...

```
XtraCommand = "System.DBAdmin.CreateUser 20"
```

This would allow any movie that logged on (including a hacker's) to create new user records, but the new users in question would not be allowed a userLevel of more than 20. (You can't create a user record with a higher level than your own current level). However, your hacker could set up his movie to create an endless series of new users, which might eventually overflow your hard disk space. This would disrupt your service. The solution I propose below should make this more difficult.

(2)Using the Movie.cfg file

The parameters set in the Multiuser.cfg are applied by default to all movies that connect to the server. However, you can configure each movie differently. In the same folder as the Multiuser Server and the Multiuser.cfg file, you will find another file named "Movie.cfg". The Movie.cfg file is provided as an example that you can copy and modify. Open it in any word

processor. It resembles the Multiuser.cfg in a number of ways. The main difference is that it is not read automatically when the server is launched.

The `connectToNetServer()` command requires a number of parameters. If you have been using the default values of the MUSyntax movie, you will have been using a command similar to the following:

```
errorCode = gMultiuserInstance.connectToNetServer("127.0.0.1", 1626,  
[#userID: "User name", #password: "Password", #movieID: "Test"])
```

What would happen if you changed the value of the `#movieID` property to "Movie"?

Try it and see. In the MUSyntax movie, choose `connectToNetServer` in the Commands dropdown list and type the word "Movie" (without the quotation marks) in the MovieID field. Now click one of the "Execute" buttons.

As far as your movie is concerned, nothing unexpected happens. However, if you look at the console of the Multiuser Server, you will see something new.

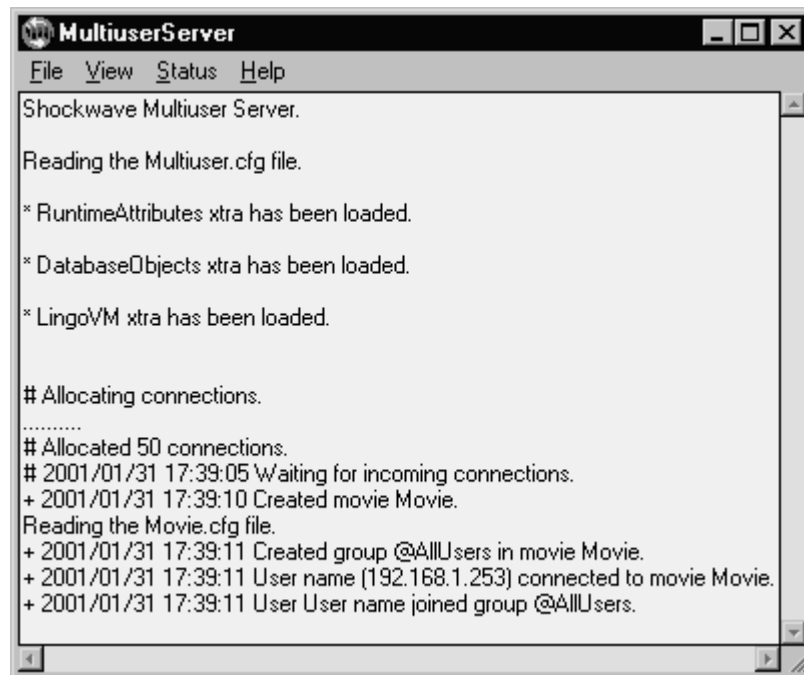


Illustration 2

Note that after the server recognizes the movie "Movie", it reads in the contents of the file "Movie.cfg" and executes any uncommented lines. In particular, when it executes the line...

```
echo Reading the Movie.cfg file.
```

... the result shows up in the server's console.

If you now connect to the server using a different `#userName` but with the same `#movieID`, the instructions in the configuration file will not be executed a second time. The server checks if any other movies with the given `#movieID` are currently running, and if not it checks if there is a configuration file associated with the new `#movieID`. As you have seen, a movie configuration file is not essential.

Since the `Movie.cfg` file is a model, it's best not to modify it in any way. Instead, make a copy of the `Movie.cfg` file and rename the new file "Subscribe.cfg". Open this new file and modify line 25 so that it now reads as follows:

```
echo Reading the Subscribe.cfg file.
```

This change is purely cosmetic. It will allow you check that the appropriate movie configuration file is being read when you connect to the server using "Subscribe" as the value for the `#movieID` property.

You can now make a much more important change. Modify line 53 so that instead of ...

```
# DefaultUserLevel = 20
```

... it now reads...

```
DefaultUserLevel = 80
```

This will give you enough authority to create a new user record. The maximum user level appears to be 100. With a user level of less than 100, you do not have access to other users' passwords. If you try working with a user level higher than 100, you may find that your requests are refused with an `errorCode` of `-2147216197: Server internal error`.

Save your changes to the `Subscribe.cfg` file, then use the `MUSyntax` movie to connect to the server using the `#movieID` "Subscribe". The connection command should look something like this:

```
errorCode = gMultiuserInstance.connectToNetServer("127.0.0.1", 1626, [#userID:  
"User name", #password: "Password", #movieID: "Subscribe"])
```

Check in the server console that the `Subscribe.cfg` file has been read.

In the `MUSyntax` movie, choose `sendNetMessage` in the `Commands` dropdown list, and use the same parameters for the `system.DBAdmin.createUser` command that you used earlier (see Figure 3).

Note:In order to gain access to your database, a hacker will need to know the value of the `#movieID` parameter that you use for your administration movies. It is thus a good idea to use an unusual `#movieID`: I use "Subscribe" here for clarity, but it is probably one of the first values that a hacker would try. A `#movieID` that resembles a cartoon swear word, such as "&#@\$!@²" is much more difficult for a hacker to guess.

(1)Unique user records

Depending on how closely you have been following, the server may either grant your request (which is encouraging), or refuse it (which shows that you've been paying attention). If the server grants your request (see Figure 4), send it again. The second time, the request should be refused (Figure 5) because you can only create one user record with a given user name.

```
errorCode = gMultiuserInstance.sendNetMessage("system.DBAdmin.createUser", "anySubject",
[#userID: "Bob", #password: "MySecret", #userLevel: 40])
put errorCode
-- 0 (No Error)

=> [#errorCode: -2147216184, #recipients: ["User name"], #senderID:
"system.DBAdmin.createUser", #subject: "anySubject", #content: [#userID: "Bob"],
#timeStamp: 7029798]
** Error: -2147216184: Data record is not unique
```

Figure 5: Only one user record is permitted for a given user name

(2)Connecting as a Registered User

We'll look more closely at the database structure in just a moment. Before that, let's see how the existence of a user record changes the log-on process.

Up to now, the server has been happy to let you log on with any userID, password and movieID. (Actually, that's not quite true: you can't use "System" as a userID, nor "Multiuser" as a movieID, since both these words have a special meaning for the server). But now that "Bob" is a registered user, you will no longer be able to log on as "Bob" unless you use the password "MySecret".

Try it. In the MUSyntax movie, choose connectToNetServer in the Commands dropdown list, and enter "Bob" as the user name and "TheWrongPassword" as the password. (The #movieID parameter can have any value). Now click on the "Execute" button. You may have to wait a while before the server will reply. A ten-second pause is built in to the server so that hackers cannot bombard it with user names and passwords in the hope of opening a connection. Eventually your movie will receive a reply with an errorCode of -2147246220, indicating that the server has refused the connection. The errorCode stands for "Invalid password". Try again using the correct password: "MySecret". This time the reply should be much faster and the errorCode should be zero.

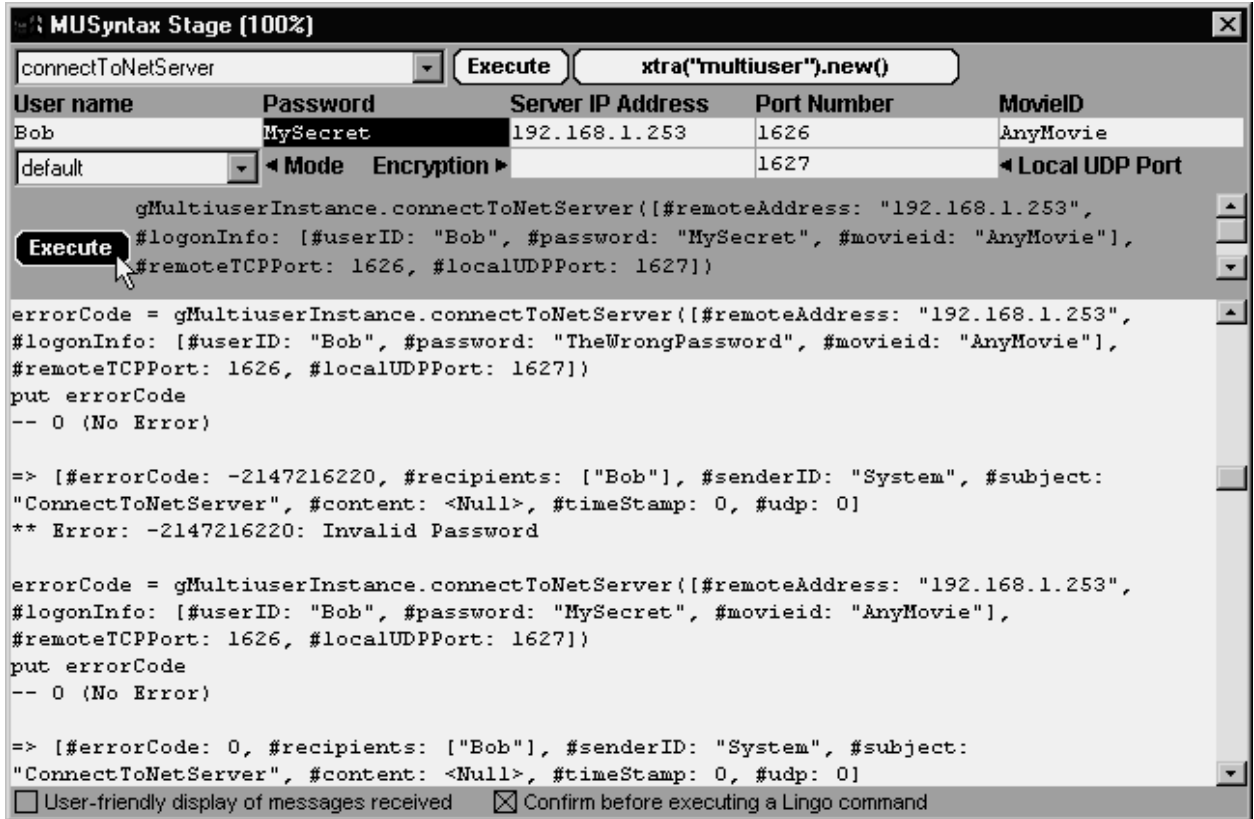


Illustration 3

(2) User Record Required

Why does the Multiuser Server now react differently? The answer lines in lines 226 - 241 of the Multiuser.cfg file:

```

#-----
# Set the authentication level that determines who can log on
# to the server. Possible values are:
#   None - anyone may log in with any name and password
#         (as long as another user has not already connected
#         to the movie using the same name) [this is the default]
#   UserRecordRequired - users may only log in if they
#         have a DBUser record in the database
#   UserRecordOptional - if a DBUser record exists for the

```

```
# the given name, then the correct password is
# required. If there is no record, then the
# connection is allowed. (user level set to the
# value of "db_default_userlevel")
#=====
XtraCommand = "Authentication UserRecordOptional"
```

You can further reinforce security by changing line 241 so that it reads:

```
XtraCommand = "Authentication UserRecordRequired"
```

Caution: If you do this, you will have to create user record for the administrative movie itself. If you did not do this, the administrative movie would not be able to log on, and could thus not create new user records.

Fortunately, the Multiuser.cfg file provides a way of creating a record automatically, as described in lines 296-309:

```
#=====
# Use the following commands to create extra user accounts
# in the database when the server starts up. These accounts
# are permanently added to the database, so future modifications
# to a username are not effective. Once that user is created, you
# could remove the line from the config file. The user level must
# be an integer value, do not use spaces. The template for
# setting this is:
# XtraCommand = "CreateUser <userName> <password> <userLevel>"
# Examples:
# XtraCommand = "CreateUser Marge Homie! 100"
# XtraCommand = "CreateUser Homer doh! 25"
# XtraCommand = "CreateUser Apu nono 80"
#=====
```

Try adding the following line to the Multiuser.cfg file at this point:

```
XtraCommand = "CreateUser AdminMovie Open5esame 80"
```

(Just to be tricky, I've used numbers instead of letters for the "O" and "S").

Save your Multiuser.cfg file, close down the server, relaunch it, and then try to log on. If you chose to set the Authentication parameter to UserRecordRequired, only two username/password pairs should now work: Bob/MySecret and AdminMovie/Open5esame. All others should be blocked. However, you can still create new users by logging on as AdminMovie. Since AdminMovie has a user level of 80, you don't specifically need to log on using the #movieID "Subscribe" to create new users. Bob, however, only has a user level of 40, so he must log on via the "Subscribe" #movieID in order to make changes.

You now have two ways of creating a new user record: you can allow new users to create their own records via an administrative movie, or you can handle the record creation yourself, via an administrative password. You could use the first system if you want your Multiuser Server to be open to the public and the second system if you want to ensure confidentiality.

(1) Inside the Database Files

New Files: DBReader.dir, DBReader.cfg, DBReader.ls

How do the files in the DBObjectFiles folder work? David Simmons, Macromedia's chief engineer on the Multiuser Server Technology, provides the following details:

The DBObject files aren't officially documented anywhere, but they are .dbf files with a FoxPro index. The DBObject database uses the Sequiter Codebase engine to create and access them.

The file structure isn't very complicated. Most are lookup tables that convert between a string (attribute name, userID, etc) to an internal ID number. Then other files contain the actual data. There's one that keeps track of the ID numbers. Here's a quick summary:

- Applications - maps between an application name and ID number
- AppData - maps between a data ID and an application ID
- AttrList - maps between an attribute name and ID
- Attributes - has info about attributes (owner ID, size, etc)

- IDTable - keeps track of the internal ID numbers so we don't get duplicates
- Players - maps user ID & application ID to a player ID
- RawData - has actual attribute data
- Users - has User's name, internal ID and password

(2)Browsing the database

You'll find a movie named "DBReader.dir" in the Multiuser\DSWMedia\DBReader\ folder on the CD-Rom that accompanies this book. This client movie requires the presence of either DBReader.cfg or DBReader.ls on the server. Both these files are available in the same folder on the CD-Rom. Each file serves a different purpose. The DBReader.cfg gives you permission to read given database files, and DBReader.ls can be used to recreate DBReader.cfg dynamically.

The DBReader.cfg file should be placed in the same folder as your Multiuser Server. It contains a list of data base files which the DBReader.dir movie may have access to. I have included the full list of standard files:

```
#=====
# General database files for the server. This is a list of
# databases associated with movies that will connect to the server.
# You may also express a path like
#           Databases = "@\db\HiScore.dbf"
# or, on a Macintosh
#           Databases = ":@db:HiScore.dbf"
# to access a path relative to the server program.
#=====
Databases = \
"@\DBObjectFiles\AppData.dbf" \
"@\DBObjectFiles\Applications.dbf" \
"@\DBObjectFiles\Attributes.dbf" \
"@\DBObjectFiles\Attrlist.dbf" \
"@\DBObjectFiles\Idtable.dbf" \
"@\DBObjectFiles\Players.dbf" \
```

```
"@\DBObjectFiles\Rawdata.dbf" \  
"@\DBObjectFiles\Users.dbf"
```

Note: The server does not treat the "@" the same way that Director does. If your server is running on a Macintosh, you should replace all the backslash (\) characters in the file name to colons (:). You will find a DBReader.cfg file written this way in the folder Multiuser\DSWMedia\DBReader\MACConfig\

If you have other files that you wish to read, you can manually add them to this list.

Remember to use the backslash character at the end of each line (except the last) to indicate to the server that the list of files continues. If the DBReader.cfg file is available, any movie that logs on with the movieID "DBReader" will be able to query any or all of the listed files.

(2) Modifying the Configuration File Dynamically

The DBReader.dir movie will work with DBReader.cfg file on its own. Server-side scripting adds power: it can be used to modify the DBReader.cfg file remotely, and thus update the list of accessible database files. But power has a price. To force the Multiuser Server to read the updated DBReader.cfg file, all movies which connect with "DBReader" as the movieID must be disconnected so that the server will delete the out-of-date "DBReader" movie object. The new values in the DBReader.cfg file will be read in when the next connection with the "DBReader" movieID is made.

If you want to be able to update the DBReader.cfg file dynamically, you will need to install the DBReader.ls script in the "Scripts" folder, and add a line to the on scriptMap handler in the Scriptmap.ls script. :

```
theMap.append([#movieID:"DBReader", #scriptFileName:"DBReader.ls"])
```

(I'll explain the Scriptmap.ls file in more detail in a later chapter). Figure 6 shows the on scriptMap handler as it may now appear:

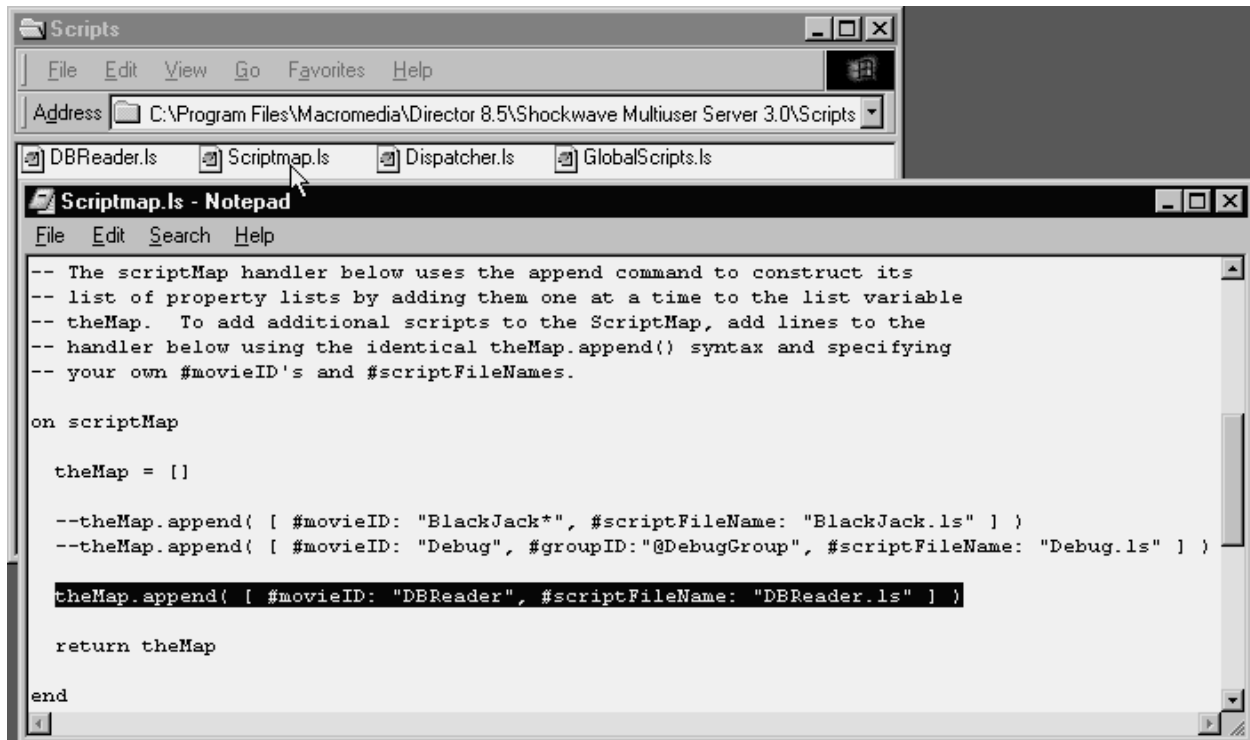


Figure 6: Enabling a server-side Lingo script

Caution: If you can read any *.dbf file using this movie, so can anyone else. The DBReader.dir movie is intended for authortime tests only. Do not leave the DBReader.cfg file next to your server when you make your server public. If you feel that you must use this technique to read database files on a public server, follow the precautions described in the DBReader.ls script. This will ensure that the only you can read the database files, and only from your own machine.

The server-side script technique can be foiled by setting the `AllowMovie` configuration parameter in the `Multiuser.cfg` file (see the "MovieID Discrimination" section on page 41 below). This will ensure that the `MultiuserServer` will not read in any modified version of the `DBReader.cfg` file: only the database files authorized by your original version of the configuration file will be accessible.

Figure 7 shows how the `DBReader` files should be placed so that the `DBReader` client movie has the greatest effect.

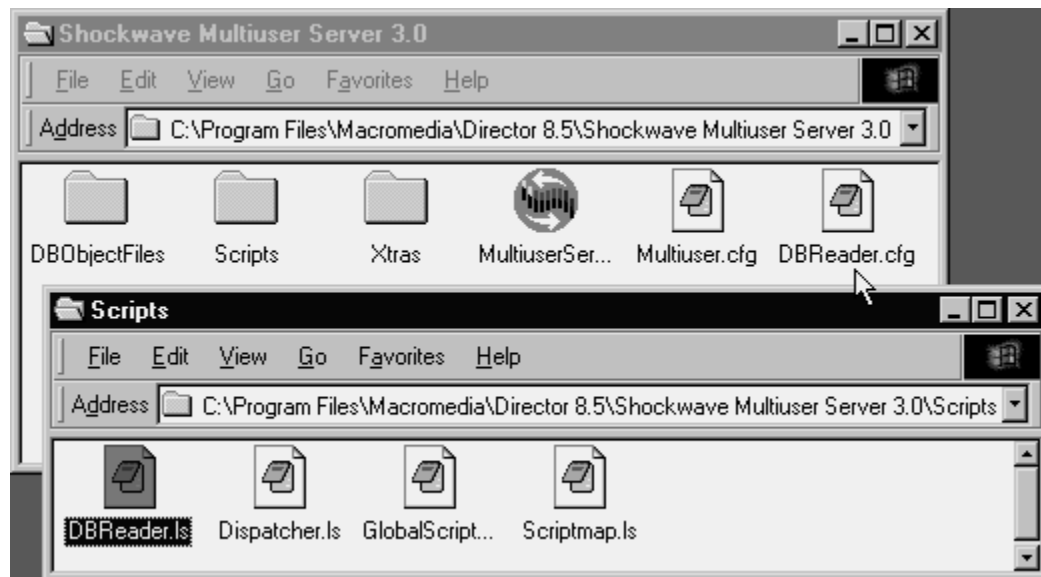


Figure 7: Where to place the `DBReader.cfg` and `DBReader.ls` files

Figure 8 shows how the `DBReader.dir` movie looks when you read the contents of the `Users.dbf` file after having created the users `Bob` and `AdminMovie`:

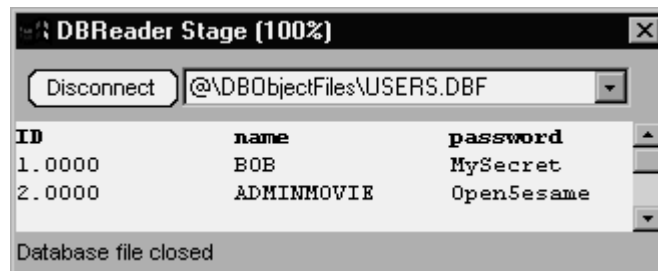
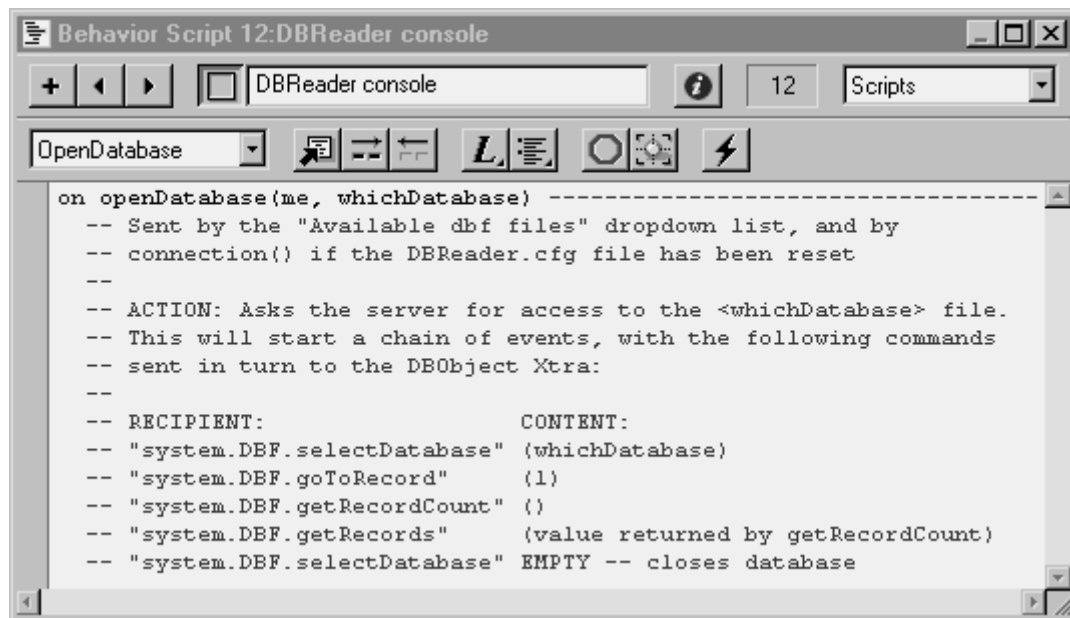


Figure 8: Reading the `Users.dbf` database file

Database files are case-sensitive. The mixed case characters for the password values indicate that passwords are case sensitive whereas the user names are not .

(2)Reading the Database Remotely

How does the DBReader.dir movie work? The main behavior "DBReader console" is attached to the text member sprite. All the other behaviors dance attendance on this one. Selecting a database file in the dropdown menu starts a series of sequence of messages between the client movie and the Multiuser Server. (To be precise, the messages are sent to the server, but they are handled DBObject Xtra). Each reply triggers a new message either until an error occurs or until the contents of the database file are read in to the client movie and the database file is closed.



```
on openDatabase(me, whichDatabase) -----
-- Sent by the "Available dbf files" dropdown list, and by
-- connection() if the DBReader.cfg file has been reset
--
-- ACTION: Asks the server for access to the <whichDatabase> file.
-- This will start a chain of events, with the following commands
-- sent in turn to the DBObject Xtra:
--
-- RECIPIENT:          CONTENT:
-- "system.DBF.selectDatabase" (whichDatabase)
-- "system.DBF.goToRecord"    (1)
-- "system.DBF.getRecordCount" {}
-- "system.DBF.getRecords"    (value returned by getRecordCount)
-- "system.DBF.selectDatabase" EMPTY -- closes database
```

Figure 9: The DBF commands sent to the server

Figure 9 shows which commands are sent to the server. These commands are described in the "Flat file database commands" section at the end of the "Using Shockwave Multiuser Server" manual. You can take a look at how these commands work in practice by studying the `on selectDatabase()` handler of the "DBReader console" behavior, plus the three handlers which follow.

To resume (for those of you who are reading this with your computer switched off): the DBReader movie asks the server to open the selected database file, and to count how many records it contains. It then asks the server to read all records from record 1 to the end. The server returns this information as a list of property lists. For the information displayed in Figure 7, this list would look like this:

```
[[#ID: 1.000, #name: "BOB", #password: "mySecret"],  
 [#ID: 2.000, #name: "ADMINMOVIE", #password: "Open5esame"]]
```

The `on convertToTable()` handler converts this list to a TAB- and RETURN-delimited string for display in the text member. In the meantime, the server closes the database file so that it can spend more time dealing with other things.

In the next section, you will be doing something similar using the following commands: `"system.DBF.selectDatabase"`, `"system.DBF.selectTag"`, `"system.DBF.seek"` and `"system.DBAdmin.createUser"`.

(1)Creating an Administration Movie

Let's now put this all into practice.

Let's imagine that a prospective client asks you to design a web site named Total Shockwave Multiuser Experience. The content hasn't been finalized yet, but your client can already provide you with the following set of notes concerning the log-on interface:

1. This is an on-line site, so you can assume that users will already be connected to the Internet.
2. First time users must register a unique log-on name and a password before entering the multuser zone.

3. The site is likely to be busy, so most popular log-on names will be taken early. We want the registration process to be as quick and easy as possible, but we do not want to make the list of existing log-on names public.
4. Once a user is registered, logging on will occur automatically on each visit to the site. However, a registered user logging on from a different machine should be given the chance of entering his or her own log-on name and password.
5. After connecting to the server, the user will see a menu screen where he or she can choose between the different games that we propose...

(2)Proof of Concept

As you will see below (or may have already noticed), this specification is both incomplete and ambiguous. Let's not worry about that yet. For the moment, let's just see how to make something like this happen. Here's one recipe in pseudo-Lingo:

```
on replyToSpec()  
  configureServerToRefuseUnregisteredUsers()  
  administrator = createUserAutomatically(with userLevel = 80)  
  
  tell the server  
    connectToNetServer(as administrator)  
    openTheDatabaseFile("Users.dbf")  
    selectTag(username column)  
  end tell  
  
  repeat while TRUE  
    tell the user  
      userName = AskTheUserForAUserName()  
      password = AskTheUserForAPassword()  
    end tell  
  end repeat  
end
```

```
end tell

tell the server
  if not thisNameIsTaken(userName) then
    closeCurrentDatabaseFile()
    newlyCreatedUser = createUser(userName, password)
    connectToNetServer(as newlyCreatedUser)
    exit repeat
  end if
end tell
end repeat

go movie "Choose your activity"
end replyToSpec
```

The pseudo-handler above gives you the sequence of events. However, talking to the server is asynchronous: you will need to send a number of messages and wait for the replies. This will mean setting up a system of callback handlers.

On the CD-Rom that accompanies this book, you will find a movie named "SeekName.dir" in the Multiuser\Register\ folder. This movie is usable but not entirely reliable. (You'll be learning from the mistakes it makes). Open the movie in Director, but don't look at any of the scripts yet. I'm assuming that your Multiuser.cfg file includes the lines ...

```
XtraCommand = "Authentication UserRecordRequired"
```

```
and
```

```
XtraCommand = "CreateUser AdminMovie Open5esame 80"
```

The first of these lines prevents unregistered users from logging on; the second creates a registered user with a userlevel of 80.

Start the SeekName.dir movie. Unless you're getting ahead of me, it should look something like this:



Illustration 4

The reason you get an error is that you have not yet provided a configuration file which gives the client movie permission to access the "Users.dbf" database file. To give your movie such permission, create a text file beside the server, call it "SeekName.cfg" and give it the following text:

```
echo Reading the SeekName.cfg file.

Databases = "@\DBObjectFiles\USERS.DBF"
```

Now try again. This time you should be more successful. The last two lines in the console should now read:

```
<= system.DBF.selectTag: No error  
=> system.DBF.selectTag: No error
```

You can now type a name into the topmost field, and click on the "system.DBF.seek" button. If all goes well, you should see an error (yes, you read that correctly):

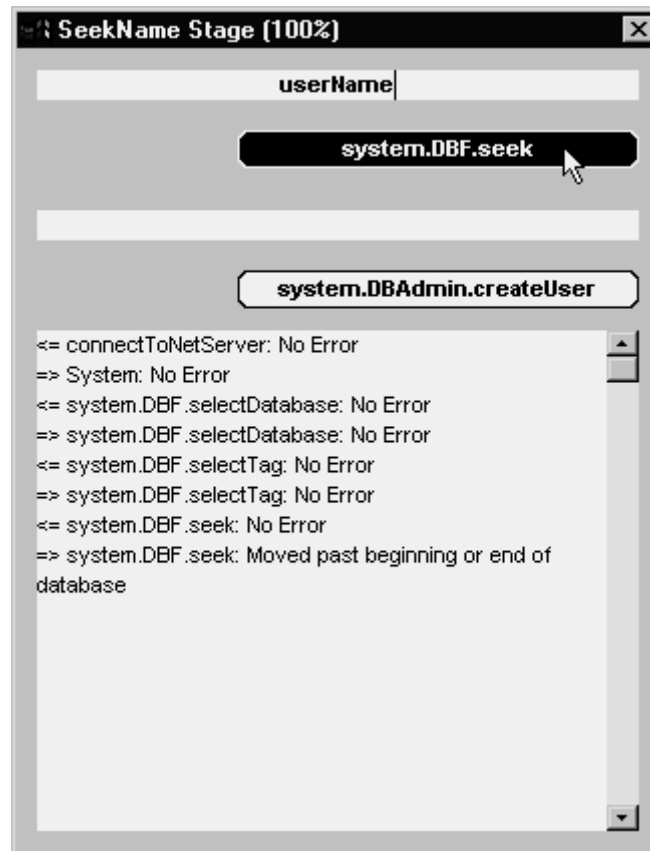


Illustration 5

This error means that the name "userName" was **not** found, which in turn means that it should be available for the new user record. You can now enter a password if you wish and click on the "system.DBAdmin.createUser" button.



Illustration 6

If all went well, you should now find yourself looking at a movie called "Lobby.dir". Your newly created user has successfully logged on.

(2) Doublechecking

You might like to open the "Users.dbf" file with the "DBReader.dir" movie, just to check that a record for "userName" was successfully entered.

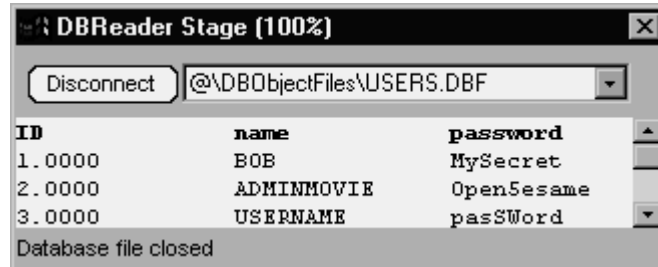


Illustration 7

Notice that `USERNAME` appears in uppercase letters, though when I entered it, only the "N" was a capital. The database is designed this way. The reason for this is that case-sensitive searching is faster than case-insensitive searching: if a letter doesn't match, a case-sensitive search will test to see if a match would occur if the letter's case were changed, and this takes time. For the same reason, the `"system.DBF.seek"` command is also case-sensitive. This has important consequences, as you shall see.

Come back to the "SeekName.dir" movie and restart it. Click on the "system.DBF.seek" to instruct the server to look for a user whose name is the mixed-case string "userName". The server complains that it can't find it. First the server told you that it had successfully created a user record with the name you provided, and now it tells you that it can't find that record.

Now type "USERNAME" all in capitals and try again. This time the "system.DBF.seek" command should find the name and you should see an alert.

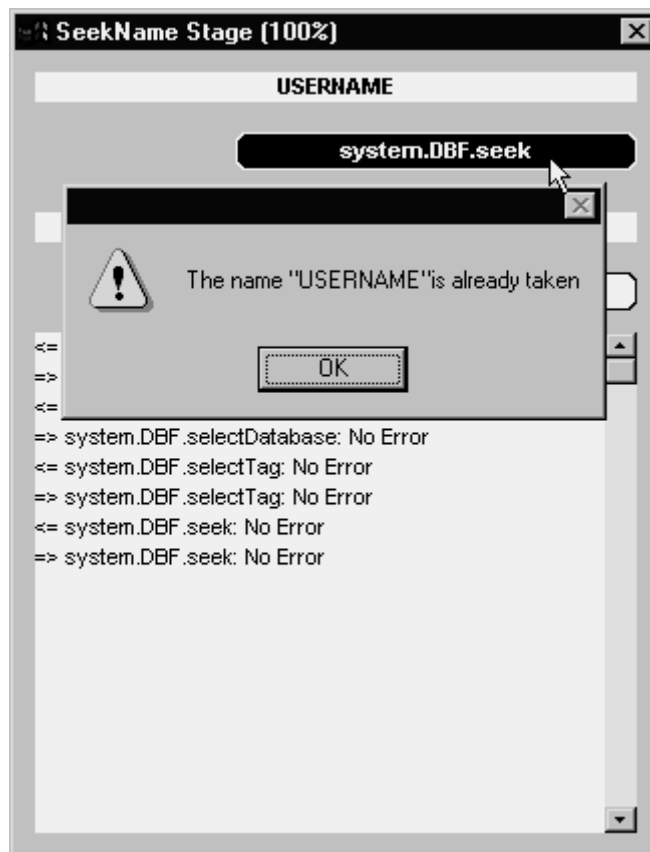


Illustration 8

The moral of this story is: you must convert a username string to uppercase before you try to find it. I told you the SeekName.dir movie was not reliable. To make it more reliable, I've included a couple of behaviors that you should drag onto the "userID" field in sprite 1: "Block RETURN", which prevents you from adding a RETURN character to the "userID" field, and "UpperCase" which acts as if the Shift-Lock key were activated.

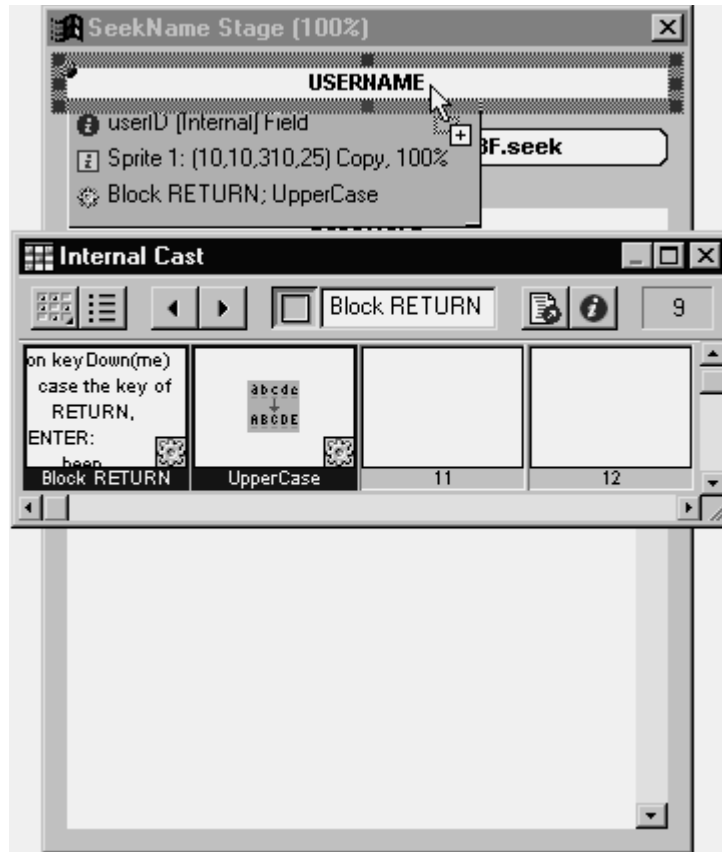


Illustration 9

(2)Writing the Script

Now that you have seen the movie in action, you should have an idea of what messages need to be sent. Open the "Seek and CreateUser" Movie Script in member 1 to see how I have implemented this. You may notice that I have used a single callback handler: `on DefaultMessageHandler()`. In most cases, a reply from the server simply generates a new command to be sent back to the server. Here are selected highlights from the handler:

```

on DefaultMessageHandler(me, incomingMessage)
  senderID = incomingMessage.senderID
  errorCode = incomingMessage.errorCode
  ShowStatus(senderID, errorCode)

  if not errorCode then
    case senderID of

```

```
"System": -- connectToNetServer
  if incomingMessage.recipients = ["AdminMovie"] then
    -- We logged on to the "SeekName" movie
    recipient = "system.DBF.selectDatabase"
    content   = "Users.dbf"
  ...
  end if
  ...
"system.DBF.selectDatabase":
  if incomingMessage.content = "Users.dbf" then
    recipient = "system.DBF.selectTag"
    content   = "NAME_TAG"
  ...
"system.DBF.selectTag":
  ...
"system.DBF.seek":
  ...
"system.DBAdmin.createUser":
  -- The new profile exists: use it to log on to the Main movie
  OpenConnection(field("userID"), field("password"), "Main")
  exit
end case

errorCode = gMUIInstance.sendNetMessage(recipient, " ", content)
ShowStatus(recipient, errorCode, TRUE)

else -- The errorCode was not zero
  if senderID = "system.DBF.seek" then
    case errorCode of
      -2147216180, -2147216181:
        -- The proposed user name was not found: we can create it.
        -- Make the "system.DBF.createUser" appear active.
        sprite(5).blendLevel = 255
    end case
  end if
end if
end DefaultMessageHandler
```

You could compare this with the pseudo-code above. If you want a challenge, you could try to fill in the gaps. If you prefer an easy solution, put a debug breakpoint at the beginning of the handler and follow the execution of the handler line by line in the Debugger window.

Note: In the DBReader.dir movie, the column containing the user names appears with the heading "name". If you use the "system.DBF.getReadableFieldList" command, the server returns the property #name. However, in the Users.dbf file, the tag associated with this column is "NAME_TAG". You must use this value in order to get the "system.DBF.selectTag" command to work.

(2)Polishing the Product

The movie you have just made simply proves that the concept is feasible, but you could hardly use it as it stands in a real project. It's time now to consider the user interface.

(3)Thinking Like an End User

In the specifications above, is there anything that the client has overlooked? Here are a couple of points that you might need to discuss with the client:

- Suppose registered user A tries to log on from registered user B's computer. How could user A do this if the connection is made automatically with user B's log on data?
- Automatic connection means that the user's log-on name and password must be recorded on a local disk. Will all users be happy about this?

Sit down with a pencil and paper, and think through the steps that the end users will need to take in order to create their log-on identity. What sort of options are they likely to want? Are these options all mutually compatible?

Can you design a series of log-on screens where the steps are simple and the options clear?

(3)Thinking Like a Developer

There could be an opportunity here to create a generic log-on module that you could re-use for other multiuser contracts. Here are a couple of points that you might discuss with the development team:

- Where does the generic module end and the client's specific content begin?
- A generic module might contain more elements than your current client wants. Could you write a generic movie that allows you to provide exactly what your client is asking for, and yet ensures that other features will be available for other projects? How much more work would this mean now? How much time could it save you later?

(3)An Example

In the Multiuser\Register\ folder of the CD-Rom that accompanies this book you will find the file "Register.htm", which launches a 22K Shockwave movie entitled "Register.dcr" when you open it in your browser. While you are considering the previous questions, you might find it helpful to test this movie and compare the way it handles with the ideal log-on dialog that you are planning to create.

If you haven't done so already, modify the "Authentication" value in line 241 of your Multiuser.cfg file so that it reads:

```
XtraCommand = "Authentication UserRecordRequired"
```

Ensure also that you have a registered user named "AdminMovie" with a userlevel of 80 and a password of "Open5esame".

Now launch the Multiuser Server on your development machine and open the "Register.htm" file in your browser.

(3)Setting the Dialog's Parameters

There are four HTML files in the same folder. Each opens the same movie (Register.dcr), using a different set of external parameters. You'll need to use a different name and password in each case. NoPass.htm doesn't require a password, OneUser.htm does. AutoLog.htm connects you to the main movie automatically, without showing you a "Connect" button. Guest.htm

allows you to log on as a guest: try using the user name and password that you set for one of the other cases. You can also try using which don't exist or the wrong password for existing names.

The differences are due to the external parameters passed to the movie from the html page it is embedded in. Open the html files in a word processor and look for the swList parameter.

For "NoPass.htm", it looks like this:

```
<object>
...
<embed>

    swList = '[#noPassword: 1, #allowGuests: 0, #preferenceFile: "NoPass"]'

</embed>

<param
    name = swList
    value = '[#noPassword: 1, #allowGuests: 0, #preferenceFile: "NoPass"]'
>
</object>
```

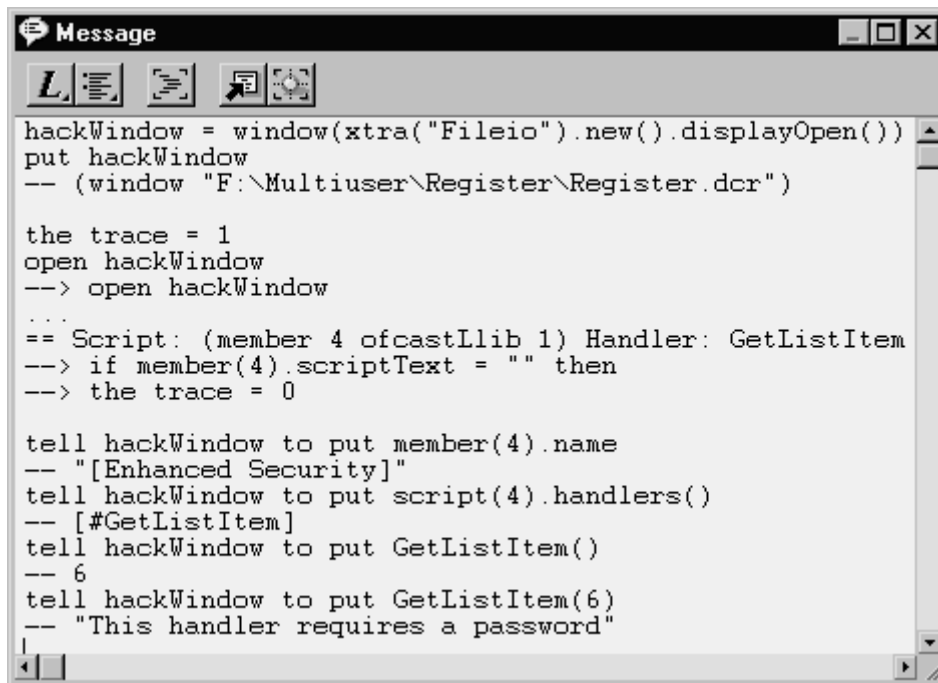
All four open the same "Lobby.dcr" movie once you have logged on, but each writes the log-on data to a different Prefs file. Once you have created a user for a given movie, press the "Reload page" button in your browser to see how the movie will appear the next time you log on.

(3)Reverse Engineering

Before you open the unprotected "Register.dir" movie, try to work out how each feature might have been implemented. Be warned: it is not nearly as simple inside as it may appear from the outside. Making the dialog user-friendly means coping with a large number of finicky details. Here are some of the features that you should note:

- If your multiuser application is popular, there may be several new users trying to register at the same time. Each will need to log on using a different administration profile; otherwise the server will refuse the connection. I provide a list of such profiles in the script of member 4. You can use the "CreateUser" XtraCommand in the Multiuser.cfg file to ensure that these administrative profiles exist.

- It is not in your interests to let hackers know what these administrative profiles are. Try opening the Shockwave movie "Register.dcr" in a window in Director's authoring environment and see if you can determine what the user name and passwords are. Below are some commands that might be helpful (but I hope not). These are some of the standard "reverse engineering" techniques that a hacker might use on your movies. I've done my best to defend my movie against them. When you open the Register.dir movie, take a look at the "[Enhanced Security]" script so that you can adapt the same tricks to your own ends.

A screenshot of a 'Message' window in Director. The window title is 'Message' and it has standard Mac OS window controls (minimize, maximize, close). Below the title bar is a toolbar with icons for list, print, copy, and paste. The main area contains a Director script. The script starts with 'hackWindow = window(xtra("Fileio").new().displayOpen())' and 'put hackWindow'. It then opens a window named 'F:\Multiuser\Register\Register.dcr'. The script sets 'the trace = 1', opens 'hackWindow', and then checks if 'member(4).scriptText = ""'. If true, it sets 'the trace = 0'. It then tells 'hackWindow' to put 'member(4).name', '[Enhanced Security]', 'script(4).handlers()', '#GetListItem', 'GetListItem()', '6', and 'GetListItem(6)'. The final line is a comment: '-- "This handler requires a password"'.

```
hackWindow = window(xtra("Fileio").new().displayOpen())
put hackWindow
-- (window "F:\Multiuser\Register\Register.dcr")

the trace = 1
open hackWindow
--> open hackWindow
...
== Script: (member 4 ofcastLlib 1) Handler: GetListItem
--> if member(4).scriptText = "" then
--> the trace = 0

tell hackWindow to put member(4).name
-- "[Enhanced Security]"
tell hackWindow to put script(4).handlers()
-- [#GetListItem]
tell hackWindow to put GetListItem()
-- 6
tell hackWindow to put GetListItem(6)
-- "This handler requires a password"
```

Illustration 10

- The "Password" behavior places bullet or asterisk characters in the password input field while you are typing. The field is however fully editable: changing the second asterisk will change the second letter of the password. If you select to save the password to the hard disk, the number of dummy characters that appear in the field is no indication of the real length of the password.
- The DBObjects server seems perfectly happy to record a username that contains a RETURN character or one that is empty: it will return an errorCode of zero. However when you use

"system.DBF.seek" to find the record you will discover that it has not been recorded as you expected. The Register" movie ensures that the userID is perfectly valid before it records it.

- If you log on as a guest, you can't choose to save the password. If you don't save your password, you can't log on automatically. The "Cascading Checkboxes" behavior determines which options should be available at any given time and enables or disables the checkboxes appropriately. This behavior is somewhat overkill for this particular application. A slimmer hardcoded version might do the job just as well. The question is whether the time spent creating a customized hardcoded feature pays off in the file size that is shaved off.
- In the same vein as the automatic disabling of irrelevant choices, certain buttons are hidden entirely if they are not needed. The parameters set in the html page determine which buttons are visible and which are not.
- Though it is not immediately evident, the movie looks after its own localization. If you run it as a movie in a window in Director and type...

```
gUserLanguage = "French"
```

... in the Message window, all the buttons names and titles will appear in French. If you run the movie on a French OS, the same thing will happen automatically. The "Localization" Movie Script checks `(the environment)[#osLanguage]` and updates the texts automatically. If you add other languages to the `GetLocalizedString()` handler, the movie will become multilingual.

- If you are an OOP purist, you may frown at my unabashed use of `sendAllSprites` in the less polished "ad hoc scripts". I figured that using `sendAllSprites` has two advantages here: it leads to less code and thus a smaller file, and it requires fewer keystrokes and head scratches than a more rigorous system would. This movie has few sprites and only a handful of inter-sprite messages are sent. The extra CPU overhead used by `sendAllSprites` will not make a noticeable difference to the reactivity of the movie.
- If you use the File > Publish menu to create a compressed version of the Register.dir movie, you will find that it weighs 28 994 bytes. My version of Register.dcr file weighs only 22 072 bytes. How did I manage that? I stripped commented text out of the "[Settings]" field, edited

the "Fontmap.txt" so that it only deals with the fonts I use, and stripped all the author-time only handlers out of the scripts. Removing handlers like `getBehaviorDescription()` is fairly innocuous. If you remove the `getPropertyDescriptionList()` handler, be sure to click the "Keep current" button when Director warns you that the property description list for your various behaviors has changed.

(3)Multiple profiles

As you saw earlier, once you have registered a user with a particular profile you are not given the option of creating a second new user. As it currently stands, the same Prefs file can host only one user profile at a time.

Look inside the Shockwave Preferences folder. Depending on your operation system, system language and boot disk, you should find this at:

- **Window 98:** C:\WINDOWS\SYSTEM\MACROMED\Shockwave 8\Prefs\
- **Window 2000:** C:\WINNT\SYSTEM32\MACROMED\Shockwave 8\Prefs\
- **MacOS:** Macintosh HD:System Folder:Extensions:Macromedia:Shockwave 8:Prefs:

(If you can't find it directly, try looking for "Shockwave 8"). Here you should find the four files names "NoPass", "OneUser", "AutoLog" and "Guests" that the Register.dcr movie has created. Open these to see what's inside: a list containing the userID and perhaps a password and an autoLogon property. It is the presence of each of these files that prevents the Register.dcr from proposing to create a new user profile. Delete one of these Prefs files and, in your browser, relaunch the html page associated with it. This time, you should be able to create a new profile.

(3)Leave your mark

Here is a challenge: can you adapt the "Register.dir" movie so that it allows you to create multiple profiles? You could add a "New Profile" button at the "Log on" marker. This button could simply jump the playback head to the "Choose name" marker. You could create a new parameter to indicate whether the button should be available or not. (You would need to provide a default value for it in the "[Settings]" field and declare it in the "Settings" Movie Script).

If multiple profiles exist, what value(s) will you save in the Prefs file?

(1) Additional Security Features

With a movie like Register.dcr protecting entry to your server, you should be able to sleep soundly. But the self-defense capabilities of the Multiuser Server don't stop there.

(2) Cryptic connections

For added security, you can also encrypt the `connectToNetServer` message. This means that even if the hacker were able to intercept the message that the administration movie sends, he would still have to crack the encryption in order to discover the value of the `#userID`, `#password` and `#movieID` parameters.

Both the movie and the Multiuser Server have to use the same encryption key. To set the key for the server, open the Multiuser.cfg file and scroll down to line 81. Remove the `#` comment character at the beginning of the line, and enter your own encryption string:

```
# Encryption key - if used, the connecting machine must use the  
# exact same key in ConnectToNetServer()  
EncryptionKey = SomeVerySecretSequenceWithoutSpaces
```

Including the `encryptionKey` parameter in the `connectToNetServer()` command can be a little tricky, since most variations on the syntax require that it come after a "mode" parameter. The mode parameter tells the Multiuser Xtra what protocol to use with the Server: if its value is 0 (zero) or `#smus` (Shockwave MultiUser Server), then it the xtra will format its outgoing messages in a way that the Multiuser Server can understand. If its value is 1 or `#text`, then the Multiuser Xtra will send messages that an IRC or SMTP text server can deal with. (This means, for example, that you can build your own email client in Director if you wish. But that is beyond the scope of this book). The difficulty is that, at the time of writing, certain variants of the `connectToNetServer()` syntax require a symbol for the mode parameter, while others require an integer. Here is a list of variations which should work (I use "EncryptionKeyString" as an example, but you should use a different string):

```
errorCode = gMultiuserInstance.connectToNetServer("User name",  
"Password", "127.0.0.1", 1626, "Test", 0, "EncryptionKeyString")  
  
errorCode = gMultiuserInstance.connectToNetServer("127.0.0.1", 1626,
```

```
[#userID: "User name", #password: "Password", #movieid: "Test"],
#smus, "EncryptionKeyString")

errorCode = gMultiuserInstance.connectToNetServer([#remoteAddress: "127.0.0.1",
#logonInfo: [#userID: "User name", #password: "Password",
#movieid: "Test"], #remoteTCPPort: "1626", #localUDPPort: "1628",
#encryptionKey: "EncryptionKeyString"])

errorCode = gMultiuserInstance.connectToNetServer([#remoteAddress: "127.0.0.1",
#logonInfo: [#userID: "User name", #password: "Password",
#movieid: "Test"], #remoteTCPPort: "1626", #localUDPPort: "1628"
#mode: 0, #encryptionKey: "EncryptionKeyString"])

errorCode = gMultiuserInstance.connectToNetServer([#remoteAddress: "127.0.0.1",
#logonInfo: [#userID: "User name", #password: "Password",
#movieid: "Test"], #remoteTCPPort: "1626", #localUDPPort: "1628",
#mode: #smus, #encryptionKey: "EncryptionKeyString"]])
```

If you decide to use encryption, it is a good idea to copy the string from your Multiuser.cfg file and paste it into your `connectToNetServer()` command, to ensure that both versions are identical. If they are not, the connection will fail, since the information decrypted by the server will be different from the information that was sent.

(2)MovieID Discrimination

The Multiuser.cfg file provides a means of allowing only movies with a trusted `movieID` to connect at lines 134-141. This means that you can prevent a hacker from using an unknown `movieID` to connect.

```
#####
# The AllowMovie value indicates a movie that can connect to
# the server.  If not set, any movie can connect.  It may
# be set multiple times to allow multiple movies.
# Continue with additional lines with a "\".
#####
AllowMovies =      YourMovieName \
AnotherMovieName
```

You can actually put several movieIDs on the same line. If you want to include a space in any movieID, put quotation marks round it. The following line...

```
AllowMovies = singleWordName "three word name"
```

... will produce output like this in the server console:

```
+ 2001/01/22 17:10:18 Movie singleWordName created.  
+ 2001/01/22 17:10:18 Movie three word name created.  
# Allocation connections..
```

The only way to modify the list of accepted movieIDs is to modify the Multiuser.cfg file. Since this will have no effect on the Multiuser Server until it is relaunched, it means that you will momentarily have to shut the server down. This means that your service will be disrupted. It is thus best to use this possibility only once you have finalized the list of movieIDs that will be using the server. (See below for a way of making this list more flexible).

Finalizing the list of movieIDs also means finalizing the associated configuration files. If you set a value for AllowMovie, the server will automatically create the required movie objects when it is launched. If there is a configuration file for a given movieID, this will be read at this time. Changing the configuration file for the movie will have no effect until the server is relaunched.

The AllowMovie value is only fully effective if you also encrypt the connection message. If you do not do so, a cunning hacker may intercept a connection message from a trusted movie and discover what movieID it uses.

(2)Enabling and Disabling Movies

The list of accepted movieIDs set by the AllowMovies value is frozen while the server is running. You can limit this list still further by using the "system.movie.disable" command. This means that you could initially provide a lengthy list of potential movieIDs in the Multiuser.cfg file, then disable those that are not currently needed. If your Multiuser.cfg file allows connections with the movieID "Test", then you can use an administrative movie to

prevent users from connecting with that `movieID`. Here is the readout from the console of the MUSyntax movie that shows this process:

```
errorCode = gMultiuserInstance.sendNetMessage("system.movie.disable",
"anySubject", "Test")
put errorCode
-- 0

=> [#errorCode: 0, #recipients: ["AdminMovie"], #senderID: "system.movie.disable",
#subject: "anySubject", #content: "Test", #timeStamp: 12145645]

errorCode = gMultiuserInstance.connectToNetServer("UserName", "Password",
"127.0.0.1", 1626, "Test")
put errorCode
-- 0 (No error)

=> [#errorCode: -2147216222, #recipients: ["UserName"], #senderID: "System",
#subject: "ConnectToNetServer", #content: <Null>, #timeStamp: 0]
** Error -2147216222: Invalid movie ID
```

If you later wish to connect to your server with a new movie, you could re-enable the movie "Test", and use "Test" as the `movieID` parameter in the `connectToNetServer()` command of your new movie.

```
errorCode = gMultiuserInstance.sendNetMessage("system.movie.enable", "anySubject",
"Test")
put errorCode
-- 0 (Aucune erreur)

=> [#errorCode: 0, #recipients: ["AdminMovie"], #senderID: "system.movie.enable",
#subject: "anySubject", #content: "Test", #timeStamp: 12630307]

errorCode = gMultiuserInstance.connectToNetServer("UserName", "Password",
"127.0.0.1", 1626, "Test")
put errorCode
-- 0 (Aucune erreur)

=> [#errorCode: 0, #recipients: ["UserName"], #senderID: "System", #subject:
"ConnectToNetServer", #content: <Null>, #timeStamp: 0]
```

The "system.movie.disable" command and the AllowMovies configuration value work independently of each other. The "system.movie.disable" command could be used on its own, for instance, to prevent users logging on to a competition movie after the deadline for last entries. Using the two techniques together gives you tight but flexible control over which movieIDs can be used to access your server and this gives hackers a harder time. The down side is that you cannot alter the configuration of a particular movie without relaunching the server.

(2)Absolute Addresses

One of the easiest ways a hacker can parasite your Multiuser Server is to copy one of your own Multiuser movies out of his browser cache and place it on his own site. By doing this, the hacker steals your intellectual property, your bandwidth and available connections on your server. You do all the work, and he takes credit for it on his site.

The solution is simple: you indicate in the Multiuser.cfg file the absolute path to the official version of each of the movies that is allowed to connect to the server. Take a look at line 167:

```
MoviePathName = http://www.mydomain.com/myMovie.dcr
```

Replace the dummy file name by the absolute path to your own movie. Note that the file extension (dir or dcr) is important. Use the continuation character to add more movie names, each on a separate line. If a path name contains a space, the whole path should be enclosed in quote marks. Make sure that you include the absolute path to any administrative movies that you might want to run.

Here's an explanation from Macromedia's David Simmons:

The MoviePathName attempts to match a string with what the client's "the moviePath" property is. The way it works is the client xtra asks Director for "the moviePath" and includes it in the logon packet sent to the server. That string must match with the one you specify in Multiuser.cfg to allow the user to log on.

Note: Users connecting to your server from a movie on their hard disk may log in with an arbitrary value for the moviePath. Fortunately, this MoviePathName check

only occurs if you use the property list format for the `connectToNetServer()` command. This means that can use this technique to protect your Shockwave movies and yet still allow users to log in from a local disk. Simply use the plain vanilla parameter format for movies which you want to accept regardless their value for the `moviePath`:

```
muInstance.connectToNetServer("userName", "password", "123.45.67.89",  
1626, "movieID")
```

(2)Trusted Domains

The Multiuser Server must know the IP address of each user that connects in order to be able to send messages back to the user. From Director 8.5 onwards, this IP address is available to the server-side scripts. The server creates a "ServerUser" object for each connected user: the ServerUser object has the following properties:

- name
- userLevel
- serverMovie
- ipAddress
- creationTime

You can thus use server-side scripting to ensure that only users from certain domains (IP addresses) can connect to a given movie. For example:

```
on userLogOn(me, movie, group, user)  
  case movie.name of  
    "RestrictedAccess": -- Add other movieIDs which use the same domains here  
      if user.ipAddress <> "123.45.67.89" then  
        put "!!! User "&user.name&" attempting to connect to \  
movie "&movie.name&" from the domain "&user.ipAddress  
  
        -- Disconnect the movie immediately  
        movie.sendMessage("system.user.delete", "", user.name)  
      end if
```

```
-- Add other movies with different parameters here
end case
end
```

You will find a script that does this in the `Multiuser\RestrictedAccess\` folder on the CD-Rom. You will need to copy the `Restricted.ls` script into the "Scripts" folder next to your server, and add a line to the `on scriptMap` handler in the `Scriptmap.ls` script:

```
theMap.append([#movieID: "*", #scriptFileName: "Restricted.ls"])
```

The asterisk (*) means that the `Restricted.ls` script's `userLogOn` handler will be called each time a user connects, regardless of the `movieID` used. You could modify this so that the `Restricted.ls` script is only used for certain movie names. See Figure 6 above for an idea of how the `Scriptmap.ls` script should look.

(2)Undocumented Feature

If users from a particular domain have been disrupting your server or one of its movies, you can use a line like the following in either the `Multiuser.cfg` or a movie configuration file.

```
BlockIPAddress = 123.45.67.*
```

This will reject connections that match the IP address. Unfortunately, you won't be able to implement this dynamically as soon as you notice interference from a given domain. You will have to halt the server or delete the movie from the server since changes to the configuration file will take effect when the configuration file is reloaded.

You can list multiple addresses or patterns to match, or have multiple lines in the configuration file. If you use a pattern, you don't need to provide a full address: `123.*` has the same effect as `123.*.45.67`. All information after the asterisk is ignored. This can be useful if you are working on an intranet, and you want to limit access to a particular group of users.

As with all undocumented features, Macromedia takes no responsibility for your use or misuse of this tag. You might find that a server-side script gives you more flexibility.

(2)A Last Word on Server Security

The default configuration for the Multiuser Server makes it easy to get started. Any user with any password and movieID can connect. Once the content on your server starts to attract attention you (and your clients) may find it important to tighten up the security aspects. By customizing the configuration and adding server-side scripting, you can make your server much more difficult to attack.

However, all this assumes that the machine your Multiuser Server is running is itself protected from intrusion. The Multiuser Server can do little to protect itself if a hacker can gain access to the files on the machine's hard disk. Discuss with your network administrator what firewall and proxy systems may be most appropriate.