The Enterprise Middleware Solution

# BEA TUXEDO

## Application Development Guide

BEA TUXEDO Release 6.5
Document Edition 6.5
February 1999

## Copyright

Copyright © 1999 BEA Systems, Inc. All Rights Reserved.

## Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

## Trademarks or Service Marks

BEA, ObjectBroker, TOP END, TUXEDO, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, Jolt and M3 are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

**BEA TUXEDO Application Development Guide**

| Document Edition | Date | Software Release |
|---|---|---|
| Version 6.5 | February 1999 | BEA TUXEDO Version 6.5 |

# Contents

## 1. A Simple Application


About This Guide ........ 1-1
Organization of the Guide ........ 1-1
Assumptions ........ 1-2
Documentation Roadmap ........ 1-2
About This Chapter ........ 1-3
Some Preliminaries ........ 1-3
The simpapp Tutorial ........ 1-4
Step 1: Copy the simpapp Files ........ 1-4
Step 2: Examine the Client Program ........ 1-5
References ........ 1-7
Step 3: Compile the Client ........ 1-8
References ........ 1-8
Step 4: Examine the Server ........ 1-8
References ........ 1-10
Step 5: Compile the Server ........ 1-10
References ........ 1-11
Step 6: Edit the Configuration File ........ 1-11
References ........ 1-12
Step 7: Load the Configuration File ........ 1-13
References ........ 1-13
Step 8: Boot the Application ........ 1-13
References ........ 1-14
Step 9: Enter a Request ........ 1-14
Step 10: Using tmadmin ........ 1-14
References ........ 1-15

Step 11: Shut Down the Application ........ 1-15
References ........ 1-16
Summary ........ 1-17

## 2. bankapp Files

Directory Structure for bankapp ........ 2-1
Files ........ 2-2
Edit bankvar to Set Environment Variables ........ 2-6
Additional PATH Component for SunOS ........ 2-10

## 3. bankapp Client Programs

A Look at bankapp Client Programs ........ 3-1
System Client Programs ........ 3-2
Mask Source Code ........ 3-3
Using mio(1) ........ 3-4
Buffer Types ........ 3-5
Using ud(1) ........ 3-5
audit.c: A Request/Response Client ........ 3-5
audit.c Source Code ........ 3-6
auditcon.c: A Conversational Client ........ 3-7
auditcon.c Source Code ........ 3-8
bankmgr.c: A Client that Monitors Events ........ 3-8
Building Client Programs ........ 3-8
References ........ 3-9

## 4. bankapp Servers

A Look at bankapp Servers ........ 4-1
Request/response Servers ........ 4-2
A Conversational Server ........ 4-3
Service Definitions ........ 4-3
Service Algorithms ........ 4-4
Utilities Incorporated into Servers ........ 4-10
Building Servers ........ 4-10
Using the buildserver Command in the bankapp ........ 4-10
The ACCT Server ........ 4-11
The BAL Server ........ 4-12

The BTADD Server ........ 4-13
The TLR Server ........ 4-13
The XFER Server ........ 4-14
Servers Built in bankapp.mk ........ 4-14
Alternative Way to Code Services ........ 4-15
References ........ 4-15

## 5. The bankapp Makefile

A Look at the bankapp Makefile ........ 5-1
Editing bankapp.mk ........ 5-1
TUXDIR ........ 5-1
APPDIR ........ 5-2
NATIVE and Other /Host Parameters ........ 5-2
Resource Manager ........ 5-3
Running bankapp.mk ........ 5-3

## 6. Databases for bankapp

Resource Manager Options for bankapp ........ 6-1
The System/D RM and bankapp ........ 6-1
Create Database in SHM Mode ........ 6-2
Create the Database in MP Mode ........ 6-2
Failure with a semget Error ........ 6-2
Using an XA-compliant RM with bankapp ........ 6-3
Changes to bankvar ........ 6-3
Changes to the bankapp Services ........ 6-3
Change to bankapp.mk ........ 6-4
Changes to crbank and crbankdb ........ 6-4
Changes to the Configuration File ........ 6-5
Using a non-XA Compliant RM with bankapp ........ 6-6
Changes to bankvar ........ 6-6
Changes to the bankapp Clients and Services ........ 6-7
Changes to bankapp.mk ........ 6-7
Changes to crbank and crbankdb ........ 6-8
Changes to the Configuration File ........ 6-8
Changes to the Driver Scripts ........ 6-8

## 7. Edit bankapp Configuration Files

Configuration Files for bankapp ............ 7-1
Notes to Listing 7-1 ............ 7-3
References ............ 7-4

## 8. Create tuxconfig, tlog; Start tlisten

Creating tuxconfig, tlog tlisten ............ 8-1
Loading the Configuration File ............ 8-1
Creating the TLOG ............ 8-2
Starting tlisten ............ 8-3
Stopping tlisten ............ 8-3
Error Messages from tlisten Problems ............ 8-4
References ............ 8-5

## 9. Boot the Application; Populate the Database

tmboot and populate ............ 9-1
Checking IPC Resources ............ 9-1
Executing tmboot ............ 9-3
The Userlog: ULOG ............ 9-3
Running the populate Script ............ 9-4
References ............ 9-4

## 10. Run bankapp

Run the Application ............ 10-1
The bankapp run Script ............ 10-1
Running the audit Client Program ............ 10-2
Running auditcon ............ 10-2
Using the driver Program ............ 10-3
Using tmadmin ............ 10-3
Shutting Down bankapp ............ 10-3
References ............ 10-4

CHAPTER

# 1 A Simple Application

## About This Guide

This is the BEA TUXEDO Application Development Guide. Its purpose is to describe how to put together a working BEA TUXEDO application so you can more easily develop applications of your own. The sample applications `simpapp` and `bankapp` come with the software. `simpapp` is described in Chapter 1 and `bankapp` is used as an example throughout the remainder of the guide.

## Organization of the Guide

The BEA TUXEDO Application Development Guide consists of the following ten chapters:

- Chapter 1 as noted above, tells how to install and run `simpapp` on your system.
- Chapter 2 lists the files that are delivered with `bankapp` and tells how to set the environment
- Chapter 3 describes the client programs of `bankapp`
- Chapter 4 describes the service subroutines of `bankapp`
- Chapter 5 describes how to edit the file `bankapp.mk` and make `bankapp`
- Chapter 6 describes how to create the database that `bankapp` was written for and how to integrate other resource managers with the system
- Chapter 7 tells how to edit the `bankapp` configuration file for your installation

- Chapter 8 describes how to load the configuration file, create the transaction log, and start the BEA TUXEDO network listener process
- Chapter 9 tells how to boot the application and populate the database
- Chapter 10 tells how to run the application

# Assumptions

We assume that readers of this guide are UNIX system users with some experience in application development, administration, or programming. We also assume some familiarity with the nature of BEA TUXEDO software, at least as much as can be gained by reading the *BEA TUXEDO Product Overview*.

An SDK license is required to build BEA TUXEDO applications.

# Documentation Roadmap

In addition to describing how to bring up and run a sample application, in this book we hope to familiarize you with the rest of the BEA TUXEDO documentation set. To that end, most chapters in this book close with a section that refers to other guides where the topics of that chapter are dealt with in more detail. In most cases, we do not think you will have to refer to other documents to bring up bankapp successfully, but when you do run into topics on which you would like more information, you can follow those pointers.

# About This Chapter

This chapter contains a tutorial that describes a simple one-client, one-server application called simpapp. An interactive form of this application is distributed with the BEA TUXEDO software.

If you follow the ten steps of the tutorial you will do the following:

- Learn how a BEA TUXEDO application is organized
- See how clients and servers are written and compiled
- Understand how an application is described in the configuration file
- Actually create an executable version of simpapp
- Boot, run, and shut down the application

## Some Preliminaries

Before you can run this tutorial the BEA TUXEDO software must be installed so that the files and commands referred to in this chapter are available.

If you are personally responsible for installing the BEA TUXEDO software, consult the *BEA TUXEDO Installation Guide* for information about how to install the BEA TUXEDO system.

If the installation has already been done by someone else, you need to know the pathname of the directory of the installed software (TUXDIR). You also need to have read and execute permissions on the directories and files in the BEA TUXEDO directory structure so you can copy simpapp files and execute BEA TUXEDO commands.

# The simpapp Tutorial

`simpapp` is a very basic BEA TUXEDO application. It has one client and one server. The server performs only one service: it accepts a string from the client and returns the same string in upper case.

The tutorial consists of ten steps designed to introduce you to the BEA TUXEDO system by showing how an application is developed and by encouraging you to bring the application up and run it. Each of the ten steps includes one or more smaller steps.

## Step 1: Copy the simpapp Files

1. Make a directory for `simpapp` and `cd` to it.

   ```
   mkdir simpdir
   cd simpdir
   ```

   This is suggested so you will be able to see clearly the `simpapp` files you have at the start and the additional files you create along the way. Use the standard shell (`/bin/sh`) or the Korn shell; not `csh`.

2. Set and export environment variables.

   ```
   TUXDIR=<pathname of the BEA TUXEDO system root directory>
   TUXCONFIG=<pathname of your present working directory>/tuxconfig
   PATH=$PATH:$TUXDIR/bin
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TUXDIR/lib
    export TUXDIR TUXCONFIG PATH LD_LIBRARY_PATH
   ```

   You need `TUXDIR` and `PATH` to be able to access files in the BEA TUXEDO system directory structure and to execute BEA TUXEDO system commands. On SunOS, `/usr/5bin` must be the first directory in your `PATH`. With AIX on the RS6000, use `LIBPATH` instead of `LD_LIBRARY_PATH`. On HPUX on the HP9000, use `SHLIB_PATH` instead of `LD_LIBRARY_PATH`.

   You need to set `TUXCONFIG` to be able to load the configuration file, which is described in “Step 7: Load the Configuration File.”

3. Copy the `simpapp` files.

   ```
   cp $TUXDIR/apps/simpapp/*.
   ```

   **Note:** Later on you will edit some of the files and make them executable, so it is best to begin with a copy of the files rather than the originals delivered with the software.

4. List the files.

   ```
   $ ls
   README         env           simpapp.nt    ubbmp       wsimpcl
   README.as400   setenv.cmd    simpcl.c      ubbsimple
   README.nt      simpapp.mk    simpserv.c    ubbws
   $
   ```

   The three files that are central to the application are:

   - `simpcl.c`—the source code for the client program
   - `simpserv.c`—the source code for the server program
   - `ubbsimple`—the ASCII form of the configuration file for the application

Except for the README files, the other files are variations of these for non-UNIX system platforms. The README files provide explanations of the other files.

# Step 2: Examine the Client Program

1. Page through the client program source code.

   ```
   $ more simpcl.c
   ```

   The output is shown in Listing 1-1.

**Listing 1-1 Source Code of simpcl.c**

```
1     #include <stdio.h>          /* UNIX */
2     #include "atmi.h"           /* TUXEDO */
3
4
5
6
7     #ifdef __STDC__
8     main(int argc, char *argv[])
```

```
9
10    #else
11
12    main(argc, argv)
13    int argc;
14    char *argv[];
15    #endif
16
17    {
18
19             char *sendbuf, *rcvbuf;
20             int sendlen, rcvlen;
21             int ret;
22
23             if(argc != 2) {
24                      fprintf(stderr, "Usage: simpcl string\n");
25                      exit(1);
26             }
27             /* Attach to System/T as a Client Process */
28             if (tpinit((TPINIT *) NULL) == -1) {
29                      fprintf(stderr, "Tpinit failed\n");
30                      exit(1);
31             }
32             sendlen = strlen(argv[1]);
33            if((sendbuf = (char *)tpalloc("STRING", NULL, sendlen+1))== NULL) {
34                      fprintf(stderr,"Error allocating send buffer\n");
35                      tpterm();
36                      exit(1);
37             }
38             if((rcvbuf = (char *)tpalloc("STRING", NULL, sendlen+1))== NULL) {
39                      fprintf(stderr,"Error allocating receive buffer\n");
40                      tpfree(sendbuf);
41                      tpterm();
42                      exit(1);
43             }
44             strcpy(sendbuf, argv[1]);
45             ret = tpcall("TOUPPER", sendbuf, NULL, &rcvbuf, &rcvlen, 0);
46             if(ret == -1) {
47                    fprintf(stderr, "Can't send request to service TOUPPER\n");
48                      fprintf(stderr, "Tperrno = %d, %s\n", tperrno,
49                                tmemsgs[tperrno]);
50                      tpfree(sendbuf);
51                      tpfree(rcvbuf);
52                      tpterm();
53                      exit(1);
54             }
55             printf("Returned string is: %s\n", rcvbuf);
56
57             /* Free Buffers & Detach from System/T */
```

```
58              tpfree(sendbuf);
59              tpfree(rcvbuf);
60              tpterm();
61    }
```

Here are eight important things to see in this file.

| | | |
|---|---|---|
| line 2 | `atmi.h` | Header file needed whenever BEA TUXEDO ATMI calls are used |
| line 28 | `tpinit()` | The ATMI call used by a client program to join an application |
| line 33 | `tpalloc()` | The ATMI call used to allocate a typed buffer. `STRING` is one of the four basic BEA TUXEDO buffer types; `NULL` indicates there is no sub-type argument. The remaining argument, `sendlen + 1`, specifies the length of the buffer plus 1 for the null character that ends the string. |
| line 38 | `tpalloc()` | Allocates another buffer for the return message |
| line 45 | `tpcall()` | Sends the message buffer to the service specified in the first argument. Also includes the address of the return buffer. `tpcall` waits for a return message. |
| lines 35, 41, 52, 60 | `tpterm()` | The ATMI call used to leave an application. A call to `tpterm()` is used to leave the application prior to taking an exit due to an error condition (lines 36, 42, and 53). The final `tpterm()` (line 60) comes after the message has been printed. |
| lines 40, 50, 51, 58, 59 | `tpfree()` | The counterpart of `tpalloc()` to free allocated buffers. |
| line 55 | `printf()` | This is the successful conclusion of the program. It prints out the message returned from the server. |

## References

The ATMI calls cited above are documented in Section 3c of the *BEA TUXEDO Reference Manual*.

# Step 3: Compile the Client

1. Run `buildclient` to compile the client program.

   ```
   buildclient -o simpcl -f simpcl.c
   ```

   where the output file is `simpcl`, and the input source file is `simpcl.c`.

2. Check the results.

   ```
   $ ls -l
   total 97
   -rwxr-x--x 1 usrid  grpid 313091   May 28 15:41  simpcl
   -rw-r----- 1 usrid  grpid   1064   May 28 07:51  simpcl.c
   -rw-r----- 1 usrid  grpid    275   May 28 08:57  simpserv.c
   -rw-r----- 1 usrid  grpid    392   May 28 07:51  ubbsimple
   ```

   As can be seen, we now have an executable module called `simpcl`. The size of `simpcl` may vary.

## References

`buildclient` is documented in `buildclient`(1).

# Step 4: Examine the Server

1. Page through the server program source code.

   ```
   $ more simpserv.c
   ```

**Listing 1-2 Source Code of simpserv.c**

```
#include <stdio.h>
     #include <ctype.h>
     #include <atmi.h>       /* TUXEDO Header File */
     #include <userlog.h>    /* TUXEDO Header File */
             /* tpsvrinit is executed when a server is booted, before it begins
                processing requests. It is not necessary to have this function.
                Also available is tpsvrdone (not used in this example), which is
                called at server shutdown time.
             */
             #if defined(__STDC__) || defined(__cplusplus)
```

```
Note 1. tpsvrinit(int argc, char *argv[])
        #else
        tpsvrinit(argc, argv)
        int argc;
        char **argv;
        #endif
        {
                /* Some compilers warn if argc and argv aren't used. */
                argc = argc;
                argv = argv;
                /* userlog writes to the central TUXEDO message log */
                userlog("Welcome to the simple server");
                return(0);
        }
       /* This function performs the actual service requested by the client.
         Its argument is a structure containing among other things a pointer
           to the data buffer, and the length of the data buffer.
        */
        #ifdef __cplusplus
        extern "C"
        #endif
        void
        #if defined(__STDC__) || defined(__cplusplus)
Note 2. TOUPPER(TPSVCINFO *rqst)
        #else
        TOUPPER(rqst)
        TPSVCINFO *rqst;
        #endif
        {
                int i;
Note 3.         for(i = 0; i < rqst->len-1; i++)
                        rqst->data[i] = toupper(rqst->data[i]);
                /* Return the transformed buffer to the requester. /

              Note 4.         tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
                      }
             #include stdio.h>
           }
```

Here are five important things to see in this file.

| | | |
|---|---|---|
| whole file | | Notice that a BEA TUXEDO server does not contain a `main()`. The `main()` is provided by the BEA TUXEDO system when the server is built. |
| Note 1 | `tpsvrinit()` | This subroutine is called during server initialization, before the server begins processing service requests. A default (provided by the BEA TUXEDO system) writes a message to `userlog` indicating that the server has been booted. `userlog`(3c) is a log that is used by the BEA TUXEDO system and can be used by applications. We will see the format in Step 10. |
| Note 2 | `TOUPPER` | The declaration of a service (the only one offered by `simpserv`). The sole argument expected by the service is a pointer to a `TPSVCINFO` structure, which contains the data string to be converted to uppercase. |
| Note 3 | `for` loop | Converts the input to uppercase by repeated calls to `toupper`. |
| Note 4 | `tpreturn` | Returns the converted string to the client with the `TPSUCCESS` flag set. |

## References

The ATMI calls and structure cited above are documented in Section 3c of the *BEA TUXEDO Reference Manual*.

# Step 5: Compile the Server

1. Run `buildserver` to compile the server program:

   ```
   buildserver -o simpserv -f simpserv.c -s TOUPPER
   ```

   where the executable file to be created is named `simpserv`, and `simpserv.c` is the input source file. The `-s TOUPPER` option specifies the service to be advertised when the server is booted.

2. Check the results.

   ```
   $ ls -l
   total 97
   -rwxr-x--x 1  usrid grpid 313091    May 28 15:41 simpcl
   -rw-r----- 1  usrid grpid   1064    May 28 07:51 simpcl.c
   -rwxr-x--x 1  usrid grpid 358369    May 29 09:00 simpserv
   -rw-r----- 1  usrid grpid    275    May 28 08:57 simpserv.c
   -rw-r----- 1  usrid grpid    392    May 28 07:51 ubbsimple
   ```

As can be seen, we now have an executable module called `simpserv`.

## References

`buildserver` is documented in `buildserver`(1).

# Step 6: Edit the Configuration File

1. Edit the file.

**Listing 1-3 The simpapp Configuration File**

```
$ vi ubbsimple

#Skeleton UBBCONFIG file for the BEA TUXEDO Simple Application.
#Replace the <bracketed> items with the appropriate values.

*RESOURCES
IPCKEY             <Replace with valid IPC Key greater than 32,768>

#Example:
#IPCKEY             62345

MASTER             simple
MAXACCESSERS       5
MAXSERVERS         5
MAXSERVICES        10
MODEL              SHM
LDBAL              N

*MACHINES

DEFAULT:
                   APPDIR="<Replace with the current pathname>"
                   TUXCONFIG="<Replace with TUXCONFIG Pathname>"
                   TUXDIR="<Root directory of TUXEDO (not /)>"
#Example:
#                  APPDIR="/usr/me/simpdir"
#                  TUXCONFIG="/usr/me/simpdir/tuxconfig"
#                  TUXDIR="/usr/tuxedo"

<Machine-name>     LMID=simple
```

```
#Example:
#tuxmach          LMID=simple

*GROUPS
GROUP1
        LMID=simple        GRPNO=1  OPENINFO=NONE

*SERVERS
DEFAULT:
                  CLOPT="-A"

simpserv          SRVGRP=GROUP1 SRVID=1

*SERVICES
TOUPPER
```

2. Change values enclosed in angle brackets to your own local values:

| | |
|---|---|
| IPCKEY | Use a value that will not conflict with any other users |
| TUXCONFIG | Provide the full pathname of the binary tuxconfig file to be created in Step 7 |
| TUXDIR | Provide the full pathname of your BEA TUXEDO root directory |
| APPDIR | Provide the full pathname of the directory where you intend to boot the application; in this case, the current directory |
| *machine-name* | Provide the machine name as returned by uname -n |

3. The pathnames for TUXCONFIG and TUXDIR must be identical to those you set and exported in Step 1.2. The strings must be the actual values; environment variables (like $TUXCONFIG, for example) are not acceptable.

   **Note:** Do not forget to remove the angle brackets.

## References

The configuration file is documented in ubbconfig(5).

# Step 7: Load the Configuration File

1. Run `tmloadcf` to load the configuration file.

```
$ tmloadcf ubbsimple
Initialize TUXCONFIG file: /usr/me/simpdir/tuxconfig [y, q] ? y
$
```

2. Check the results.

```
$ ls -l
total 216
-rwxr-x--x 1 usrid grpid 313091    May 28 15:41 simpcl
-rw-r----- 1 usrid grpid   1064    May 28 07:51 simpcl.c
-rwxr-x--x 1 usrid grpid 358369    May 29 09:00 simpserv
-rw-r----- 1 usrid grpid    275    May 28 08:57 simpserv.c
-rw-r----- 1 usrid grpid 106496    May 29 09:27 tuxconfig
-rw-r----- 1 usrid grpid    382    May 29 09:26 ubbsimple
```

We see that we now have a file called `tuxconfig`. The `tuxconfig` file is a new file system under the control of the BEA TUXEDO system.

## References

`tmloadcf` is documented in `tmloadcf`(1).

# Step 8: Boot the Application

1. Execute `tmboot` to bring up the application.

```
$ tmboot
Boot all admin and server processes? (y/n): y
Booting all admin and server processes in
/usr/me/simpdir/tuxconfig

Booting all admin processes ...

exec BBL -A:
      process id=24223 ... Started.

Booting server processes ...

exec simpserv -A :
      process id=24257 ... Started.
2 processes started.
$
```

BBL is the administrative process that monitors the application shared memory structures. `simpserv` is our server that runs continuously awaiting requests.

### References

tmboot is documented in tmboot(1).

## Step 9: Enter a Request

1. Run the client program to submit a request.

```
$ simpcl "hello, world"
Returned string is: HELLO, WORLD
```

We are successful!

## Step 10: Using tmadmin

tmadmin is an interactive program that an administrator can use to check an application and make dynamic changes. It requires the TUXCONFIG variable to be set. We will show you just two of the many tmadmin commands.

1. Enter the following command.

```
$ tmadmin
```

   You will see the following lines.

```
tmadmin - Copyright (c) 1998 BEA Systems, Inc. All rights
reserved.

>
```

   The greater-than sign (>) is the tmadmin prompt.

2. Enter the printserver(psr) command to display information about the servers.

```
> psr
a.out Name Queue Name  Grp Name ID RqDone Load Done Current Service
---------- ----------  -------- -- ------ --------- ---------------
BBL        531993      simple    0      0         0 ( IDLE )
simpserv   00001.00001 GROUP1    1      0         0 ( IDLE )
>
```

3. Enter the `printservice(psc)` command to display information about the services:

```
> psc
Service Name Routine Name a.out Name Grp Name ID Machine # Done Status
------------ ------------ ---------- -------- -- ------- ------ ------
TOUPPER      TOUPPER      simpserv   GROUP1    1 simple       - AVAIL
>
```

4. Leave `tmadmin` by entering a `q` at the prompt. You can boot and shut down the application from within `tmadmin`. We have done those functions with shell commands in Step 8 and Step 11, respectively.

## References

`tmadmin` is documented in `tmadmin`(1).

# Step 11: Shut Down the Application

1. Run `tmshutdown` to bring the application down.

```
$ tmshutdown
Shutdown all admin and server processes? (y/n): y
Shutting down all admin and server processes in
/usr/me/simpdir/tuxconfig

Shutting down server processes ...

  Server Id = 1 Group Id = GROUP1 Machine = simple:    shutdown
succeeded.

Shutting down admin processes ...

  Server Id = 0 Group Id = simple Machine = simple:    shutdown
succeeded.
2 processes stopped.
$
```

2. Check the `ULOG`.

```
$ cat ULOG*
$
113837.tuxmach!tmloadcf.10261: CMDTUX_CAT:879:
       A new file system has been created. (size = 32 4096-byte blocks)
113842.tuxmach!tmloadcf.10261: CMDTUX_CAT:871:
       TUXCONFIG file /usr/me/simpdir/tuxconfig has been created
113908.tuxmach!BBL.10768: LIBTUX_CAT:262: std main starting
113913.tuxmach!simpserv.10925: LIBTUX_CAT:262: std main starting
113913.tuxmach!simpserv.10925: Welcome to the simple server
114009.tuxmach!simpserv.10925: LIBTUX_CAT:522:
       Default tpsvrdone() function used.
114012.tuxmach!BBL.10768: CMDTUX_CAT:26: Exiting system
```

Each line of the `ULOG` for this session contains something of interest. Most are self-explanatory, but we want to add some explanation for a couple of them. First let's look at the format of a `ULOG` line.

```
time (hhmmss).machine_uname!process_name.process_id: log message
```

Now let's look at some individual lines.

```
113913.Message from tpsvrinit() in simpserv
114009.When simpserv is shutdown the BEA TUXEDO main sends this message
```

## References

`tmshutdown` is documented in `tmshutdown`(1).

The userlog is documented in `userlog`(3c).

# Summary

If you have reached this point, you have successfully brought up, run, and brought down a BEA TUXEDO system application. You have seen what a client program and a server look like. You have edited a configuration file to refer to your own environment. You have invoked `tmadmin` to check on the activity of your application. In all the applications you may work on in the future the basic elements of client processes, server processes, and a configuration file will be present, and you will have all of the BEA TUXEDO shell commands at your fingertips.

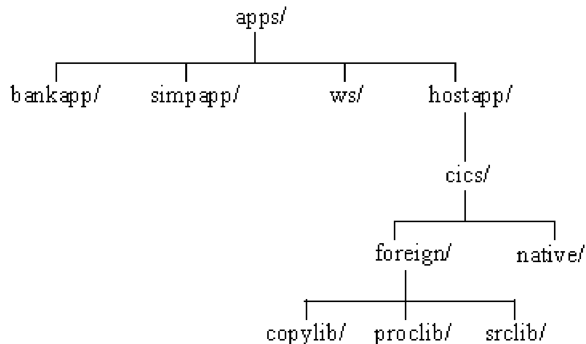
Good luck!

# 2 bankapp Files

## Directory Structure for bankapp

This chapter describes the directory structure under the `apps` directory, which is subordinate to the root directory for your BEA TUXEDO system software. We will also take a look at the files in the `bankapp` directory. The directory structure is shown in Figure 2-1.

**Figure 2-1 Directory Structure under apps/**




`simpapp` is described in Chapter 1, "A Simple Application." `hostapp` is not distributed except under special arrangements.

# Files

Table 2-1 lists the files of the banking application. The left hand column lists the source files delivered with the BEA TUXEDO software. The center column lists files that are generated when the `bankapp.mk` is run. The right hand column gives a brief summary of the purpose of the file.

**Table 2-1 Banking Application Files**

| Source | Generated | Purpose |
|---|---|---|
| `ACCT.ec` | `ACCT.c`, `ACCT.o`, `ACCT` | Contains `OPEN_ACCT` and `CLOSE_ACCT` services to open and close accounts. |
| `ACCTMGR.c` | `ACCTMGR` | A server that subscribes to events and logs notifications. Contains `WATCHDOG` and `Q_OPENACCT_LOG` services. |
| `AUDITC.c` | `AUDITC` | Contains a conversational server that handles service requests from the client `auditcon` |
| `BAL.ec` | `BAL.c`, `BAL.o`, `BAL` | Contains `ABAL`, `TBAL`, `ABAL_BID` and `TBAL_BID` services to allow the audit client to obtain bank-wide or branch-wide account or teller balances. |
| `BALC.ec` | `BALC.c BALC.o BALC` | Contains `ABALC_BID`, and `TBALC_BID`. These services are the same as `TBAL_BID` and `ABAL_BID` above, except that `TPSUCCESS` is returned when a branch id is not found. This allows `auditcon` to continue. |
| `BALANCE.m` | `BALANCE.M` | Mask for balance inquiry data entry. |
| `bankmgr.c` | `bankmgr` | A client program that subscribes to events of special interest. |
| `BTADD.ec` | `BTADD.c`, `BTADD.o`, `BTADD` | Contains `BR_ADD` and `TLR_ADD` services to allow addition of branches or tellers to the database. |
| `CBALANCE.m` | `CBALANCE.M` | Mask for confirmation of a balance inquiry. |
| `CCLOSE.m` | `CCLOSE.M` | Mask for confirmation of an account closing. |
| `CDEPOSIT.m` | `CDEPOSIT.M` | Mask for confirmation of a deposit. |
| `CLOSE.m` | `CLOSE.M` | Mask for account closing data entry. |
| `COPEN.m` | `COPEN.M` | Mask for confirmation of an account opening. |

**Table 2-1 Banking Application Files**

| Source | Generated | Purpose |
|---|---|---|
| cracl.sh | – | A shell script that creates Access Control Lists to demonstrate the Access Control security level. |
| crqueue.sh | – | A shell script that creates application queues for use in event notification. |
| crusers.sh | – | A shell script that creates groups and users to demonstrate the authentication security level. |
| CTRANSFER.m | CTRANSFER.M | Mask for confirmation of a transfer. |
| CWITHDRAW.m | CWITHDRAW.M | Mask for confirmation of a withdrawal. |
| DEPOSIT.m | DEPOSIT.M | Mask for deposit data entry. |
| event.flds | – | A field table file used in the event feature. |
| FILES | – | Descriptive list of all the files in bankapp. |
| HELP.m | HELP.M | Mask that explains mio keystrokes. |
| MENU.m | MENU.M | Mask that offers ring menu to choose deposit, withdrawal, transfer, balance inquiry, open account, or close account data entry screens. |
| OPEN.m | OPEN.M | Mask for open account data entry. |
| README | – | Installation and boot procedures. |
| README.nt | – | Installation and boot procedures for the NT platform. |
| README2 | – | Documentation of additions to bankapp that demonstrate new features. The file is located in the apps/bankapp directory. |
| README2.nt | – | Documentation of additions to bankapp that demonstrate new features for the NT platform. The file is located in the apps/bankapp directory. |
| RUNME.sh | – | Interactive script to build, configure, boot, shutdown application. |
| showq.sh! | – | A shell script that displays the status and contents of a message queue. |
| TLR.ec | TLR.c, TLR.o, TLR | Contains WITHDRAWAL, DEPOSIT and INQUIRY services. |
| TRANSFER.m | TRANSFER.M | Mask for transfer data entry. |
| usrevtf.sh | – | Creates an ENVFILE for the BEA TUXEDO server TMUSREVT. |

**Table 2-1 Banking Application Files**

| Source | Generated | Purpose |
|---|---|---|
| WITHDRAW.m | WITHDRAW.M | Mask for withdrawal data entry. |
| XFER.c | XFER.o, XFER | Contains TRANSFER service. |
| aud.v | aud.V, aud.h | FML view used to define structure passed between audit client and the BAL server. |
| appinit.c | appinit.o | Contains tpsvrinit() and tpsvrdone() for all servers other than TLR. |
| audit.c | audit.o, audit | Client that obtains bank-wide or branch-wide account and teller balances via the ABAL, TBAL, ABAL_BID and TBAL_BID services. |
| auditcon.c | auditcon | interactive version of audit that uses conversations and services ABAL, TBAL, ABALC_BID, TBALC_BID. |
| bankapp.mk | – | Application makefile. |
| bankapp.nt | – | Application makefile for NT. |
| bank.flds | bank.flds.h | Field table file containing bank database fields and auxiliary FML fields used by masks and servers. |
| bank.h | – | Contains data definitions pertinent to more than just one C program within the application. |
| bankvar | – | Contains variable settings, except for those within ENVFILE. Because it sets ENVFILE itself, setting bankvar will set the entire environment. |
| crbank.sh | crbank | Creates databases for all banks when using SHM mode. See Chapter 1, “A Simple Application,” for guidelines on use. |
| crbankdb.sh | crbankdb | Creates a database for one server group. See Chapter 1, “A Simple Application,” for guidelines on use. |
| crtlog.sh | crtlog, TLOG | Creates a UDL and a TLOG on the master site. Creates a UDL on the non-master sites. tmboot creates a TLOG on the non-master sites. |
| driver.sh | driver | Drives the application by piping FML buffers with transaction requests through ud(1). |
| envfile.sh | envfile, ENVFILE | Creates ENVFILE for use by tmloadcf. |

**Table 2-1 Banking Application Files**

| Source | Generated | Purpose |
|---|---|---|
| gendata.c | gendata | Generates ud-readable requests to add ten branches, thirty tellers and two hundred accounts. |
| gentran.c | gentran | Generates ud-readable transaction requests from among DEPOSIT, WITHDRAWAL, TRANSFER and INQUIRY. |
| populate.sh | populate | Populates the database by piping FML buffers with branch, teller and account add requests through ud(1). |
| run.sh | run | Invokes mio with MENU mask. |
| ubbmp | tuxconfig | Sample UBBCONFIG file for use in a MP mode configuration. |
| ubbshm | tuxconfig | Sample UBBCONFIG file for use in a SHM mode configuration. |
| util.c | util.o | Contains a function commonly used among all services, namely getstr(). |

Of the forty odd files in the directory:

- 14 are .m files that create data entry masks managed by the system client program, mio(1).
- 5 are .ec files that are source files for service subroutines using embedded SQL statements.
- 8 are .c files; audit.c is a client program; auditcon.c is a conversational client that connects to AUDITC.c, which is a conversational server; three others are servers or associated with servers, two are there to generate data or transactions for the application.

The remaining files have various roles; some are files you need in any application, others are make files for various add-ons, still others are present simply to facilitate the use of bankapp as an example. In subsequent chapters we will closely examine a number of the files, and give a more complete explanation of their role in the sample application. For now we just want to discuss the bankvar file.

# Edit bankvar to Set Environment Variables

bankvar is a file of environment variables needed by bankapp. The file bankvar is approximately 185 lines due largely to the extensive comments, but there are only a few that you should be concerned about immediately.

The first key line checks to see if TUXDIR is set. If it is not, execution of the file fails with the message:

```
TUXDIR: parameter null or not set
```

So, set TUXDIR to the root directory of your BEA TUXEDO system directory structure, and export it.

Another line in bankvar sets APPDIR to the directory ${TUXDIR}/apps/bankapp, which is the directory where bankapp source files are located. APPDIR is a directory where BEA TUXEDO looks for your application-specific files. You might prefer to copy the bankapp files to a different directory to safeguard the original source files. If you do, then the directory you use should be entered here. It does not have to be under TUXDIR.

Another important line sets a value for DIPCKEY. This is an IPCKEY for a BEA TUXEDO system database. There is a discussion of databases in Chapter 6; the use of this key is described there. For now, all you need to know about it is that it must be different from the value of the BEA TUXEDO IPCKEY specified in the UBBCONFIG file (Chapter 7).

The other variables specified in bankvar play various roles in the sample application and you will need to be aware of them when you are developing your own application. They will all be mentioned at appropriate places later in this guide. Grouping them all in bankvar is done to show you an example that you may want to adapt at a later time for use with a real application.

When you have made all the changes to bankvar that you need to, execute bankvar as follows:

```
. ./bankvar
```

**Listing 2-1 bankvar: Environment Variables for bankapp**

```
#Copyright (c) 1997, 1996 BEA Systems, Inc.
#Copyright (c) 1995, 1994 Novell, Inc.
#Copyright (c) 1993, 1992, 1991, 1990 Unix System Laboratories, Inc.
#All rights reserved
#
# This file sets all the environment variables needed by the BEA TUXEDO software
# to run the bankapp
#
# This directory contains all the BEA TUXEDO software
# System administrator must set this variable
#
if [ -z "${TUXDIR}" ] ; then
if [ ! -z "${ROOTDIR}" ] ; then
TUXDIR=$ROOTDIR
export TUXDIR
fi
fi
TUXDIR=${TUXDIR:?}
#
# This directory contains all the user written code
#
# Contains the full path name of the directory that the application
# generator should place the files it creates
#
APPDIR=${TUXDIR}/apps/bankapp
#
# This path contains the shared objects that are dynamically linked at
# runtime in certain environments, e.g., SVR4.
#
LD_LIBRARY_PATH=${TUXDIR}/lib:${LD_LIBRARY_PATH}
#
# Logical block size; Database Administrator must set this variable
#
BLKSIZE=512
#
# Set default name of the database to be used by database utilities
# and database creation scripts
#
DBNAME=bankdb
#
# Indicate whether database is to be opened in share or private mode
#
DBPRIVATE=no
#
# Set Ipc Key for the database; this MUST differ from the UBBCONFIG
# *RESOURCES IPCKEY parameter
```

```
#
DIPCKEY=80953
#
# Environment file to be used by tmloadcf
#
ENVFILE=${APPDIR}/ENVFILE
#
# List of field table files to be used by mc, viewc, tmloadcf, etc.
#
FIELDTBLS=Usysflds,bank.flds,credit.flds,event.flds
#
FIELDTBLS32=Usysfl32,evt_mib,tpadm
#
# List of directories to search to find field table files
#
FLDTBLDIR=${TUXDIR}/udataobj:${APPDIR}
#
FLDTBLDIR32=${TUXDIR}/udataobj:${APPDIR}
#
# Universal Device List for database
#
FSCONFIG=${APPDIR}/bankdl1
#
# List of directories to search to find mask files for mio
#
MASKPATH=${APPDIR}
#
# Network address, used in MENU script
#
NADDR=
#
# Network device name
#
NDEVICE=
#
# Network listener address, used in MENU script
#
NLSADDR=
#
# List of services permitted to the current invoker of mio
#
OKXACTS=ALL
#
# Make sure TERM is set for mio
#
TERM=${TERM:?}
#
# Set device for the transaction log; this should match the TLOGDEVICE
# parameter under this site's LMID in the *MACHINES section of the
```

```
# UBBCONFIG file
#
TLOGDEVICE=${APPDIR}/TLOG
#
# Device for binary file that gives /T all its information
#
TUXCONFIG=${APPDIR}/tuxconfig
#
# Set the prefix of the file which is to contain the central user log;
# this should match the ULOGPFX parameter under this site's LMID in the
# *MACHINES section of the UBBCONFIG file
#
ULOGPFX=${APPDIR}/ULOG
#
# System name, used by RUNME.sh
#
UNAME=
#
# List of view files to be used by viewc, tmloadcf, etc.
#
VIEWFILES=aud.V
#
VIEWFILES32=mib_views,tmib_views
#
# List of directories to search to find view files
#
VIEWDIR=${TUXDIR}/udataobj:${APPDIR}
#
VIEWDIR32=${TUXDIR}/udataobj:${APPDIR}
#
# Specify the Q device (if events included in demo)
#
QMCONFIG=${APPDIR}/qdevice
#
# Export all variables just set
#
export TUXDIR APPDIR BLKSIZE DBNAME DBPRIVATE DIPCKEY ENVFILE
export LD_LIBRARY_PATH
export FIELDTBLS FLDTBLDIR FSCONFIG MASKPATH OKXACTS TERM
export FIELDTBLS32 FLDTBLDIR32
export TLOGDEVICE TUXCONFIG ULOGPFX
export VIEWDIR VIEWFILES
export VIEWDIR32 VIEWFILES32
export QMCONFIG
#
# Add TUXDIR/bin to PATH if not already there
#
a="`echo $PATH | grep ${TUXDIR}/bin`"
if [ x"$a" = x ]
```

```
then
PATH=${TUXDIR}/bin:${PATH}
export PATH
fi
#
# Add APPDIR to PATH if not already there
#
a="`echo $PATH | grep ${APPDIR}`"
if [ x"$a" = x ]
then
PATH=${PATH}:${APPDIR}
export PATH
fi
#
# Check for other machine types bin directories
#
for DIR in /usr/5bin /usr/ccs/bin /opt/SUNWspro/bin
do
if [ -d ${DIR} ] ; then
PATH="${DIR}:${PATH}"
fi
done
```

## Additional PATH Component for SunOS

If your operating system is SunOS, you need to put `/usr/5bin` at the front of your `PATH`. The following command can be used:

```
PATH=/usr/5bin:$PATH;export PATH
```

Another requirement for SunOS users: use `/bin/sh` rather than `csh` for your shell.

# 3 bankapp Client Programs

## A Look at bankapp Client Programs

This chapter is devoted to the client side of the `bankapp` sample application.

In the client-server architecture of BEA TUXEDO there are two modes of communication:

- Request/response mode, which is characterized by the sending of a single request for a service to be performed by the server and getting back a single response.
- Conversational mode; in this mode a dedicated connection is established between a client (or a server acting like a client) and a server. The connection remains active until terminated. While the connection is active, messages containing service requests and responses can be sent and received between the two participating processes.

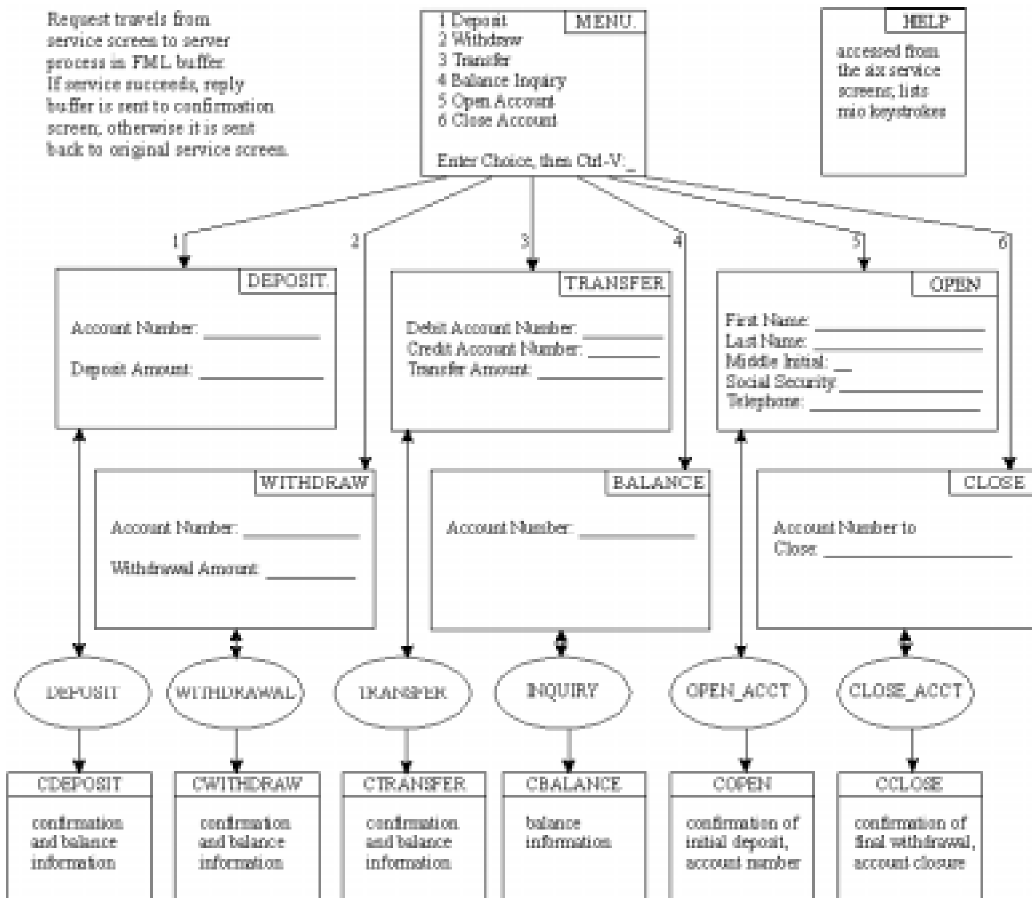
Variations of the two modes above can be constructed by taking advantage of the BEA TUXEDO features that allow requests to be forwarded from one server to another, that permit requests to be chained and that permit requests to be queued in stable storage for later processing. `bankapp` is not set up to demonstrate any of these variations, but once you have the application running you might want to try these as extensions to the example.

# System Client Programs

One form of client access to `bankapp` is through the resources of the BEA TUXEDO Data Entry System (DES), a character-oriented interface. With DES, data entry forms (also called masks or screens) are created to provide a template that can be used by application users to formulate requests. The masks can be organized into a hierarchy by means of `MENU` statements of the form definition language, `UFORM`. They are managed by the system client, `mio`(1).

Figure 3-1 shows the hierarchy of masks for `bankapp`. The top-level mask is a menu that leads the user to select one of the six service request masks. The oval shapes in the illustration represent application services. The six rectangles across the bottom of Figure 3-1 represent confirmation masks that give feedback about the results of the service request.

**Figure 3-1 The bankapp Input/Output Mask Hierarchy**




## Mask Source Code

Taking one of the shorter masks for illustration, in Listing 3-1 we show how the source code of a mask looks in the `UFORM` syntax. This mask (indicated as number 6 in Figure 3-1) is used to close an account. It calls the `CLOSE_ACCT` service and has a single variable field for the number of the account to be closed.

Once a mask has been created, it is converted into binary form and is used under the control of `mio`.

**Listing 3-1 Source Code for the CLOSE.m Mask**

```
#
#
#SERVICE NAME=CLOSE_ACCT
#FORM FLAGS=Umrv TRANMODE=TRAN TRANTIME=30
#PAGE STATUSLINE=24 FLAGS=Pmrv
*ROW    COL    MIN    LINES    WIDTH    FLAGS    VALUE
*---    ---    ---    -----    -----    -----    -----
2       C      -      1        -        L        "TUXEDO (R) System"
+1      C      -      1        -        L        "Banking Services"
+2      C      -      1        -        L        "Close Account"
+6      25     -      1        -        L        "Account Number To Close:"
-       51     5      1        7        UmN7IHrv  ACCOUNT_ID
HELP="Enter account number"
ERR="Account number must be 7 digit number"
VAL=IR:[1-9999999]
FORMEXIT    F0=FC:HELP,F11=S:CLOSE_ACCT
+3      24    -      1         -        L          "Hit CTRL-v to complete trans."
+1       C     -     1         -        L          "or ESC 0 for keystroke help"
```

## Using mio(1)

mio(1) is a forms handling program supplied by the BEA TUXEDO system that gathers the data from a binary data entry mask into a buffer and sends the buffer to a service. bankapp has a set of masks (shown above in Figure 3-1) that mio uses for calling the OPEN_ACCT, CLOSE_ACCT, WITHDRAWAL, DEPOSIT, INQUIRY, and TRANSFER services. mio joins the application as a client and when the user enters the key sequence to transmit the mask, the BEA TUXEDO software adds the service request to the queue of a server that advertises the desired service. If the application is using an application password, mio prompts the user to enter the password before allowing any of the service request screens to be used.

If mio is invoked with no arguments, it presents a generic initial mask that prompts the user to name the mask to bring up. In bankapp, the shell script named run invokes mio with the initial menu for bankapp. If you look at run.sh, you will see that it contains one command line:

```
mio -i MENU
```

Of course, you can also get into the mask system by invoking mio directly rather than through run.

## Buffer Types

It was mentioned in the preceding section that mio gathers the data from a data entry mask into a buffer before sending it to a service. Message buffers are an essential part of BEA TUXEDO, as is the concept of typed buffers. In BEA TUXEDO a typed buffer is a buffer designed to hold a specific data type. Nine types are defined: FML, FML32, VIEW, VIEW32, STRING and CARRAY plus three versions for X/OPEN compatibility. Applications have the ability to define additional types. An FML buffer is a fielded buffer in which each field carries its own identifying information. mio and other BEA TUXEDO client programs use FML buffers.

## Using ud(1)

Another system client program used by bankapp is ud(1). ud is supplied by the BEA TUXEDO System to allow fielded buffers to be read from standard input and sent to a service. In the sample application, ud is used by both the populate and driver programs. In populate, a program called gendata passes service requests to ud with customer account information to be entered in the bankapp database; in driver, the data flow is similar, but the program is gentran and the purpose is to throw transactions at the application to simulate an active system.

# audit.c: A Request/Response Client

audit.c is an example of a client program that does not use the BEA TUXEDO DES. It makes branch-wide or bank-wide balance inquiries that call on the services ABAL, TBAL, ABAL_BID and TBAL_BID. As an executable, it is invoked in one of two ways:

`audit [-a | -t]`

Prints the bank-wide total value of all accounts, or bank-wide cash supply of all tellers. Option -a or -t must be specified to control whether account balances or teller balances are to be tallied.

`audit [-a | -t} branch_ID`

Prints branch-wide total value of all accounts, or branch-wide cash supply of all tellers, for branch denoted by *branch_ID*. Option -a or -t must be specified to control whether account balances or teller balances are to be tallied.

The algorithm for the program is shown in Listing 3-2.

**Listing 3-2 Audit Algorithm**

```
main()
{
     Parse command line options with getopt();
     Join application with tpinit();
     Begin global transaction with tpbegin();
     If (branch_id specified) {
              Allocate buffer for service requests with tpalloc();
               Place branch_id into the aud structure;
               Do tpcall() to "ABAL_BID" or "TBAL_BID";
               Print balance for branch_id;
               Free buffer with tpfree();
     }
     else /* branch_id not specified */
               call subroutine sum_bal();
     Commit global transaction with tpcommit();
     Leave application with tpterm();
}
sum_bal()
}
     Allocate buffer for service requests with tpalloc();
     For (each of several representative branch_id's,
               one for each site)
               Do tpacall() to "ABAL" or "TBAL";
     For (each representative branch_id) {
               Do tpgetrply() wtith TPGETANY flag set
                        to retrieve replies;
               Add balance to total;
               Print total balance;
     }
     Free buffer with tpfree();
}
```

## audit.c Source Code

Because of space constraints we are not going to print the entire source code of `audit.c`, but we want to call your attention to the following sections.

In the program's `main()`:

```
/* Join application */
/* Start global transaction */
/* Create buffer and set data pointer */
/* Do tpcall */
```

```
/* Commit global transaction */
/* Leave application /*
```

In the subroutine sum_bal:

```
/* Create buffer and set data pointer */
/* Do tpacall */
/* Do tpgetrplys to retrieve answers to questions */
```

The indicated sections contain all of the places in audit.c where BEA TUXEDO ATMI calls are used. Note also that audit.c is an example of a program that uses a VIEW typed buffer and a structure that is defined in the aud.h header file. The source code for the structure can be found in the view description file, aud.v.

# auditcon.c: A Conversational Client

auditcon.c is the source code for a conversational version of audit.c. After the client is built, the program is started when a user enters auditcon.

The algorithm for the program is shown in Listing 3-3.

**Listing 3-3 Algorithm for Conversational Audit**

```
main()
{
        Join the application
        Begin a transaction
        Open a connection to conversational service AUDITC
        Do until user says to quit: {
                 Query user for input
                 Send service request
                 Receive response
                 Print response on user's terminal
                 Prompt for further input
        }
        Commit transaction
        Leave the application
}
```

### auditcon.c Source Code

The source code for auditcon uses the ATMI calls for conversational communication: tpconnect(), to establish the connection between the client and service, tpsend(), to send a message, and tprecv() to receive a message.

## bankmgr.c: A Client that Monitors Events

bankmgr.c is included with bankapp as a demonstration of a client that is designed to run constantly. It subscribes to application-defined events of special interest such as the opening of a new account or a withdrawal above $10,000.

## Building Client Programs

DES masks must be compiled before they can be used by mio. If the mask is created using vuform(1), the BEA TUXEDO visual form editor, it is automatically converted to binary format (indicated by an .M suffix). If it is created by editing a file of UFORM statements, the file must be run through the BEA TUXEDO mask compiler, mc(1), which also creates an .M file. Masks created with vuform should be unloaded to ASCII .m files for backup. This was formerly done with mcdis(1).

View description files, of which aud.v is an example, are processed by the view compiler. viewc(1). viewc has two output files: a binary view description file, aud.V, and a header file, aud.h.

The client programs, audit.c and audconv.c are processed by buildclient(1) to compile them and/or link edit them with the necessary BEA TUXEDO libraries.

You can use any of these commands individually, if you choose, but rules for all these steps are included in bankapp.mk.

# References

The use of ATMI calls in client programs is covered in the *BEA TUXEDO Programmer's Guide*.

The creation of masks, the operation of `mio` and a tutorial on `vuform` are all included in the *BEA TUXEDO Data Entry System Guide*.

The subject of typed buffers is covered in both the *BEA TUXEDO Programmer's Guide* and the *Administering the BEA TUXEDO System*.

All commands and ATMI calls are described in Sections 1 and 3c of the *BEA TUXEDO Reference Manual*. The `bankmgr.c` client is more fully described in the `README2` file of `bankapp` and in the `bankmgr.c` code itself. The Event Broker/Monitor feature, which is what `bankmgr.c` demonstrates, is described in *Administering the BEA TUXEDO System*.

CHAPTER

# 4 bankapp Servers

## A Look at bankapp Servers

This chapter describes the servers delivered with `bankapp`, identifies the services coded for the banking application and describes how the services are link edited into servers.

Servers are executable processes that offer one or more services. In the BEA TUXEDO system, they continually accept requests (from processes acting as clients) and dispatch them to the appropriate services. Services are subroutines of C language code written specifically for an application. It is the services accessing a resource manager that provide the functionality for which your BEA TUXEDO system transaction processing application is being developed. Service routines are one part of the application that must be written by the BEA TUXEDO system programmer (user-defined clients being another part).

All the services in `bankapp` are coded in the C language with embedded SQL except for the `TRANSFER` service, which does not directly interact with the database. The `TRANSFER` service is offered by the `XFER` server and is a C program (that is, its source file is a `.c` file rather than a `.ec` file).

All the services of `bankapp` use functions provided in the Application Transaction Management Interface (ATMI). These functions allow the services:

- To manage typed buffers
- To communicate synchronously or asynchronously with other services
- To define global transactions
- To generically access a resource manager
- To send replies back to clients

This chapter provides the following:

- A description of each server and service that is part of the banking application
- The pseudo-code for each service that is either accessed by the BEA TUXEDO system predefined client, `mio`, or the application client, `audit`
- The relationships between the `bankapp` services and servers
- The `buildserver`(1) command options used to compile and build each server with the BEA TUXEDO system predefined `main( )`
- An alternative way to structure the same servers

# Request/response Servers

Five of the `bankapp` servers operate in request/response mode. Four of the five use embedded SQL statements to access the resource manager; in the source files in `TUXDIR/apps/bankapp` they are the files with a `.ec` suffix. The fifth server, `XFER`, for transfer, makes no calls to the resource manager itself; it calls the `WITHDRAWAL` and `DEPOSIT` services (which are offered by the `TLR` server) to transfer funds between accounts. The source file for `XFER` is a `.c` file, since `XFER` makes no resource manager calls and contains no embedded SQL statements.

`BTADD.ec`
: Allows branch and teller records to be added to the proper database from any site.

`ACCT.ec`
: Provides customer representative services, namely the opening and closing of accounts (`OPEN_ACCT` and `CLOSE_ACCT`).

`TLR.ec`
: Provides teller services, namely `WITHDRAWAL`, `DEPOSIT`, and `INQUIRY`. Each `TLR` process identifies itself as an actual teller in the `TELLER` file, via the user-defined `-T` option on the server's command line.

`XFER.c`
: Provides fund transfers for accounts anywhere in the database.

`BAL.ec`
: Sums teller or account balances for all branches of the database or for a specific branch identifier.

# A Conversational Server

The server AUDITC.c is an example of a conversational server. It has one service, which is also called AUDITC. The conversational client, auditcon, establishes a connection to AUDITC and sends it requests for audit information. AUDITC evaluates the requests and calls an appropriate service (ABAL, TBAL, ABAL_BID, or TBAL_BID) to get the information. When a reply is received from the service called, AUDITC sends it back to auditcon. An important point to observe here is that a service in a conversational server can make calls to request/response services. It can also initiate connections to other conversational servers, but that is not part of this example.

# Service Definitions

There are 12 request/response services in bankapp. Each bankapp service matches a C function name in the source code of a server, as shown in the following list.

BR_ADD
: Adds a new branch record; offered by the BTADD server; accepts an FML buffer as input.

TLR_ADD
: Adds a new teller record; offered by BTADD; accepts an FML buffer as input.

OPEN_ACCT
: Inserts a record into the ACCOUNT file and calls DEPOSIT to add the initial balance; offered by ACCT; accepts an FML buffer as input; chooses ACCOUNT_ID for a new account based on BRANCH_ID of the teller involved.

CLOSE_ACCT
: Deletes an ACCOUNT record; offered by ACCT; accepts an FML buffer as input; validates ACCOUNT_ID, calls WITHDRAWAL to remove the final balance.

WITHDRAWAL
: Subtracts an amount from the specified branch, teller and account balance; offered by TLR; accepts an FML buffer as input; validates the ACCOUNT_ID and SAMOUNT fields; checks that funds are available from account and teller.

DEPOSIT
: Adds an amount to specified branch, teller and account balances; offered by TLR; accepts an FML buffer as input, validates the ACCOUNT_ID and SAMOUNT fields.

INQUIRY
: Retrieves an account balance; offered by TLR; accepts an FML buffer as input, validates ACCOUNT_ID.

TRANSFER
: Issues a tpcall() requesting WITHDRAWAL followed by one requesting DEPOSIT; offered by XFER; accepts an FML buffer as input.

ABAL
: Sums account balances for all branches on a given site; offered by BAL; accepts the VIEW buffer of aud.v as input.

TBAL
: Sums the teller balances for all branches on a given site; offered by BAL; accepts the VIEW buffer of aud.v as input.

ABAL_BID
: Sums the account balances for a specific BRANCH_ID; offered by BAL; accepts the VIEW buffer of aud.v as input.

TBAL_BID
: Sums the teller balances for a specific BRANCH_ID; offered by BAL; accepts the VIEW buffer of aud.v as input.

# Service Algorithms

The twelve figures that follow illustrate in pseudo-code the algorithms used in the BR_ADD, TLR_ADD, OPEN_ACCT, CLOSE_ACCT, WITHDRAWAL, DEPOSIT, INQUIRY, TRANSFER, ABAL, TBAL, ABAL_BID, and TBAL_BID services. You can use them as roadmaps through the source code that can be found in servers in TUXDIR/apps/bankapp.

**Listing 4-1 The BR_ADD Algorithm**

```
void BR_ADD (TPSVCINFO *transb)
{
        set pointer to TPSVCINFO data buffer;
        get all values for service request from field buffer;
        insert record into BRANCH;
        tpreturn() with success;
}
```

**Listing 4-2 The TLR_ADD Algorithm**

```
void TLR_ADD (TPSVCINFO *transb)
{
        set pointer to TPSVCINFO data buffer;
        get all values for service request from fielded buffer;
        get TELLER_ID by reading branch's LAST_ACCT;
                 insert teller record;
        update BRANCH with new LAST_TELLER;
        tpreturn() with success;
}
```

**Listing 4-3 The OPEN_ACCT Algorithm**

```
void OPEN_ACCT(TPSVCINFO *transb)
{
   Extract all values for service request from fielded buffer using Fget() and
Fvall();
   Check that initial deposit is positive amount and tpreturn() with failure if
not;
   Check that branch id is a legal value and tpreturn() with failure if it is not;
   Set transaction consistency level to read/write;
   Retrieve BRANCH record to choose new account based on branch's LAST_ACCT field;
   Insert new account record into ACCOUNT file;
   Update BRANCH record with new value for LAST_ACCT;
   Create deposit request buffer with tpalloc(); initialize it for FML with
Finit();
   Fill deposit buffer with values for DEPOSIT service request;
   Increase priority of coming DEPOSIT request since call is from a service;
   Do tpcall() to DEPOSIT service to add amount of initial balance;
   Prepare return buffer with necessary information;
```

```
   Free deposit request buffer with tpfree();
   tpreturn() with success;
}
```

**Listing 4-4 The CLOSE_ACCT Algorithm**

```
void CLOSE_ACCT(TPSVCINFO *transb)
{
   Extract account id from fielded buffer using Fvall();
   Check that account id is a legal value and tpreturn() with failure if it is not;
   Set transaction consistency level to read/write;
   Retrieve ACCOUNT record to determine amount of final withdrawal;
   Create withdrawal request buffer with tpalloc(); initialize it for FML with
Finit();
   Fill withdrawal buffer with values for WITHDRAWAL service request;
   Increase priority of coming WITHDRAWAL request since call is from a service;
   Do tpcall() to WITHDRAWAL service to withdraw balance of account;
   Delete ACCOUNT record;
   Prepare return buffer with necessary information;
   Free withdrawal request buffer with tpfree();
   tpreturn with success;
}
```

**Listing 4-5 The WITHDRAWAL Algorithm**

```
void WITHDRAWAL(TPSVCINFO *transb)
{
   Extract account id and amount from fielded buffer using Fvall() and Fget();
   Check that account id is a legal value and tpreturn() with failure if not;
   Check that withdraw amount (amt) is positive and tpreturn() with failure if not;
   Set transaction consistency level to read/write;
   Retrieve ACCOUNT record to get account balance;
   Check that amount of withdrawal does not exceed ACCOUNT balance;
   Retrieve TELLER record to get teller's balance and branch id;
   Check that amount of withdrawal does not exceed TELLER balance;
   Retrieve BRANCH record to get branch balance;
   Check that amount of withdrawal does not exceed BRANCH balance;
   Subtract amt to obtain new account balance;
   Update ACCOUNT record with new account balance;
   Subtract amt to obtain new teller balance;
   Update TELLER record with new teller balance;
   Subtract amt to obtain new branch balance;
```

```
   Update BRANCH record with new branch balance;
   Insert new HISTORY record with transaction information;
   Prepare return buffer with necessary information;
   tpreturn with success;
}
```

**Listing 4-6 The DEPOSIT Algorithm**

```
void DEPOSIT(TPSVCINFO *transb)
{
   Extract account id and amount from fielded buffer using Fvall() and Fget();
   Check that account id is a legal value and tpreturn() with failure if not;
  Check that deposit amount (amt) is positive and tpreturn() with failure if not;
   Set transaction consistency level to read/write;
   Retrieve ACCOUNT record to get account balance;
   Retrieve TELLER record to get teller's balance and branch id;
   Retrieve BRANCH record to get branch balance;
   Add amt to obtain new account balance;
   Update ACCOUNT record with new account balance;
   Add amt to obtain new teller balance;
   Update TELLER record with new teller balance;
   Add amt to obtain new branch balance;
   Update BRANCH record with new branch balance;
   Insert new HISTORY record with transaction information;
   Prepare return buffer with necessary information;
   tpreturn() with success;
}
```

**Listing 4-7 The INQUIRY Algorithm**

```
void INQUIRY(TPSVCINFO *transb)
{
   Extract account id from fielded buffer using Fvall();
   Check that account id is a legal value and tpreturn() with failure if not;
   Set transaction consistency level to read only;
   Retrieve ACCOUNT record to get account balance;
   Prepare return buffer with necessary information;
   tpreturn() with success;
}
```

**Listing 4-8 The TRANSFER Algorithm**

```
void TRANSFER(TPSVCINFO *transb)
{
   Extract account id's and amount from fielded buffer using Fvall() and Fget();
   Check that both account ids are legal values and tpreturn() with failure if not;
   Check that transfer amount is positive and tpreturn() with failure if it is not;
   Create withdrawal request buffer with tpalloc(); initialize it for FML with
Finit();
   Fill withdrawal request buffer with values for WITHDRAWAL service request;
   Increase priority of coming WITHDRAWAL request since call is from a service;
   Do tpcall() to WITHDRAWAL service;
   Get information from returned request buffer;
   Reinitialize withdrawal request buffer for use as deposit request buffer with
Finit();
   Fill deposit request buffer with values for DEPOSIT service request;
   Increase priority of coming DEPOSIT request;
   Do tpcall() to DEPOSIT service;
   Prepare return buffer with necessary information;
   Free withdrawal/deposit request buffer with tpfree();
   tpreturn() with success;
}
```

**Listing 4-9 The ABAL Algorithm**

```
void ABAL(TPSVCINFO *transb)
{
   Set transaction consistency level to read only;
   Retrieve sum of all ACCOUNT file BALANCE values for the
     database of this server group (A single ESQL
     statement is sufficient);
   Place sum into return buffer data structure;
   tpreturn( ) with success;
}
```

**Listing 4-10 The TBAL Algorithm**

```
void TBAL(TPSVCINFO *transb)
{
   Set transaction consistency level to read only;
   Retrieve sum of all TELLER file BALANCE values for the
     database of this server group (A single ESQL
     statement is sufficient);
   Place sum into return buffer data structure;
   tpreturn( ) with success;
}
```

**Listing 4-11 The ABAL_BID Algorithm**

```
void ABAL_BID(TPSVCINFO *transb)
{
   Set transaction consistency level to read only;
   Set branch_id based on transb buffer;
   Retrieve sum of all ACCOUNT file BALANCE values for records
     having BRANCH_ID = branch_id (A single ESQL
     statement is sufficient);
   Place sum into return buffer data structure;
   tpreturn( ) with success;
}
```

**Listing 4-12 The TBAL_BID Algorithm**

```
void TBAL_BID(TPSVCINFO *transb)
{
   Set transaction consistency level to read only;
   Set branch_id based on transb buffer;
   Retrieve sum of all TELLER file BALANCE values for records
     having BRANCH_ID = branch_id (A single ESQL
     statement is sufficient);
   Place sum into return buffer data structure;
   tpreturn( ) with success;
}
```

# Utilities Incorporated into Servers

There are two C language subroutines included among the source files of `bankapp`: `appinit.c` and `util.c`.

`appinit.c` contains application-specific versions of `tpsvrinit()` and `tpsvrdone()` subroutines. `tpsvrinit()` and `tpsvrdone()` are subroutines that are included in the standard BEA TUXEDO system `main()`. The default version of `tpsvrinit()` calls `tpopen()` to open the resource manager and `userlog()` to post a message that the server has started. The default version of `tpsvrdone()` calls `tpclose()` to close the resource manager and `userlog()` to post a message that the server is about to shut down. Any application subroutines named `tpsvrinit()` and `tpsvrdone()` are used in place of the defaults, thus enabling the application to provide initialization and pre-shutdown procedures of its own.

`util.c` contains a subroutine called `getstr()`, which is used in `bankapp` to process SQL error messages.

# Building Servers

`buildserver`(1) is used to put together an executable server built on the BEA TUXEDO system's `main()`. Options identify the names of the output file, the input files provided by the application, and various libraries that permit you to run a BEA TUXEDO system application in a variety of ways.

`buildserver` invokes the `cc` command. The environment variables `CC` and `CFLAGS` can be set to name an alternative compile command and to set flags for the compile and link edit phases. The key `buildserver` command line options are illustrated in the examples that follow.

# Using the buildserver Command in the bankapp

This section provides the `buildserver` command used in `bankapp.mk` to compile and build each server in the banking application. Refer to the *BEA TUXEDO Programmer's Guide* and the `buildserver`(1) reference page in the *BEA TUXEDO Reference Manual* for complete details.

## The ACCT Server

The ACCT server is derived from an ACCT.ec file that contains the code for the OPEN_ACCT and CLOSE_ACCT functions. The ACCT.ec is first compiled to an ACCT.o file before supplying it to the buildserver command so that any compile-time errors can be clearly identified and dealt with before this step. The ACCT.o file is created in the following two steps (done for you in bankapp.mk).

1. The .c file is generated as follows.

```
esql ACCT.ec
```

2. The .o file is generated as follows.

```
cc -I $TUXDIR/include -c ACCT.c
```

The ACCT server was created by running the following buildserver command line.

```
buildserver -r TUXEDO/SQL \
            -s OPEN_ACCT -s CLOSE_ACCT \
            -o ACCT \
            -f ACCT.o -f appinit.o -f util.o
```

The explanation of the command line options is as follows:

- The -r option is used to specify which resource manager access libraries should be link edited with the executable server. The choice is specified with the strings TUXEDO/D or TUXEDO/SQL. Only one string can be specified.
- The -s option is used to specify the service names in the server that are available to be advertised when the server is booted. If the name of the function that performs a service is different from the service name, the function name becomes part of the argument of the -s option. In the bankapp, the function name is the same as the name of the service so only the service names themselves need to be specified. It is our convention to specify all uppercase for the service name. For example, the OPEN_ACCT service would be processed by function OPEN_ACCT(). However, the -s option of buildserver does allow you to specify an arbitrary name for the processing function for a service within a server. Refer to the buildserver(1) reference page for details. It is also possible for the administrator to specify that only a subset of the services that were used to create the server with the buildserver command is to be available when the server is booted. Refer to the *Administering the BEA TUXEDO System*.
- The -o option is used to assign a name to the executable output file. If no name is provided, the file is named SERVER.

- The -f option specifies the files that are used in the link edit phase. Also refer to the description of the -l option on the buildserver(1) reference page. The *BEA TUXEDO Programmer's Guide* describes both of these options in some detail as well. The order in which the files are listed is significant. The order is dependent on function references and in what libraries the references are resolved. Source modules should be listed ahead of libraries that might be used to resolve their references. If these are .c files, they are first compiled. (In the example above, appinit.o and util.o have been compiled previously.) Object files can be either separate .o files or groups of files in archive (.a) files. If more than a single file name is given as an argument to a -f option, the syntax calls for a list enclosed in double quotes. You can use as many -f options as you need.

As you can see in the previous example, the -r option was used to specify the BEA TUXEDO system SQL resource manager. The -s option names the OPEN_ACCT and CLOSE_ACCT services (which are defined by functions of the same name in the ACCT.ec file) to be the services that make up the ACCT server. The -o option assigns the name ACCT to the executable output file and the -f option specifies that the ACCT.o, appinit.o, and util.o files are to be used in the link edit phase of the build. Note that the appinit.c file contains the system supplied tpsvrinit() and tpsvrdone(). Refer to the *BEA TUXEDO Programmer's Guide* and the tpservice(3c) reference page in the *BEA TUXEDO Reference Manual* for an explanation of how these routines are used. The util.c file contains a few other commonly used routines.

## The BAL Server

The BAL server is derived from a BAL.ec file that contains the code for the ABAL, TBAL, ABAL_BID, and TBAL_BID functions. As with the ACCT.ec, the BAL.ec is first compiled to a BAL.o file before being supplied to the buildserver command for the same reasons already stated. The buildserver command that was used to build the BAL server follows:

```
buildserver -r TUXEDO/SQL \
            -s ABAL -s TBAL -s ABAL_BID -s TBAL_BID\
            -o BAL \
            -f BAL.o -f appinit.o
```

The -r option specifies the BEA TUXEDO system SQL resource manager, the -s option names the services that make up the BAL server (as before, the functions in the BAL.ec file that define these services have identical names), the -o option assigns the name BAL to the executable server, and the -f option specifies that the BAL.o and the appinit.o files are to be used in the link edit phase.

## The BTADD Server

The BTADD server is derived from a BTADD.ec file that contains the code for the BR_ADD and TLR_ADD functions. The BTADD.ec is also compiled to a BTADD.o file before being supplied to the buildserver command. The buildserver command that was used to build the BTADD server follows:

```
buildserver -r TUXEDO/SQL \
            -s BR_ADD -s TLR_ADD \
            -o BTADD \
            -f BTADD.o -f appinit.o
```

The -r option specifies the BEA TUXEDO system SQL resource manager, the -s option names the services (BR_ADD and TLR_ADD) that make up the BTADD server (the functions in the BTADD.ec file that define these services have identical names), the -o option assigns the name BTADD to the executable server, and the -f option specifies that the BTADD.o and the appinit.o files are to be used in the link edit phase.

## The TLR Server

The TLR server is derived from a TLR.ec file that contains the code for the DEPOSIT, WITHDRAWAL, and INQUIRY functions. The TLR.ec is also compiled to a TLR.o file before being supplied to the buildserver command. The buildserver command that was used to build the TLR server follows:

```
buildserver -r TUXEDO/SQL \
            -s DEPOSIT -s WITHDRAWAL -s INQUIRY \
            -o TLR \
            -f TLR.o -f util.o -f -lm
```

The -r option specifies the BEA TUXEDO system SQL resource manager, the -s option names DEPOSIT, WITHDRAWAL, and INQUIRY as the services that make up the TLR server (the functions in the TLR.ec file that define these services have identical names), the -o option assigns the name TLR to the executable server, and the -f option specifies that the TLR.o and the util.o files are to be used in the link edit phase.

Note the special use of the -f option in the previous example. In this example the -f option is also used to pass an option (-lm) to the cc command line. As stated earlier, buildserver invokes the cc command. By supplying the -lm string to the -f option, it is passed to the cc command and is then interpreted as the option that causes the math libraries to be linked in during the compilation process. Refer to the cc(1) reference page in the *UNIX System V User's Reference Manual* for a complete list of compile-time options.

## The XFER Server

The XFER server is derived from an XFER.c file that contains the code for the TRANSFER function. The XFER.c is also compiled to an XFER.o file before being supplied to the buildserver command. The buildserver command that was used to build the XFER server follows:

```
buildserver -r TUXEDO/SQL \
            -s TRANSFER \
            -o XFER \
            -f XFER.o -f appinit.o
```

The -r option specifies the BEA TUXEDO system SQL resource manager, the -s option names TRANSFER as the only service that makes up the XFER server (the function in the XFER.c file that defines the TRANSFER service has the identical name), the -o option assigns the name XFER to the executable server, and the -f option specifies that the XFER.o and the appinit.o files are to be used in the link edit phase.

## Servers Built in bankapp.mk

The preceding sections on building the bankapp servers were included because it is important that you understand how the buildserver command is specified. However, in actual practice you are apt to incorporate the build into a makefile; that is the way it is done in bankapp. The bankapp makefile is discussed in Chapter 5, “The bankapp Makefile.”

# Alternative Way to Code Services

You may have noticed that in the bankapp source files all the services were incorporated into files that we have been referring to as the source code for servers. These files do indeed have the same names as the bankapp servers, but they are not really servers. Why? Because they do not contain a main() section. A standard main() is provided by the BEA TUXEDO system at buildserver time.

An alternative organization for a BEA TUXEDO system application might be to keep each service subroutine in its individual file. We will use the TLR.ec file as an example. TLR.ec contains three services that could have been in their own separate .ec files called, for example, INQUIRY.ec, WITHDRAW.ec, and DEPOSIT.ec. The .ecs for each service would be compiled to their corresponding .os and the buildserver command line would look like the following:

```
buildserver -r TUXEDO/SQL \
            -s DEPOSIT -s WITHDRAWAL -s INQUIRY \
            -o TLR \
            -f DEPOSIT.o -f WITHDRAW.o -f INQUIRY.o \
            -f util.o -f -lm
```

As the preceding example illustrates, there is no need to code the service functions in one source file that represents the server. That is, the server does not need to have an existence as a source program file at all. It can be derived from various source files and come into existence as a server executable through the files specified on the buildserver command line. This may permit greater flexibility in building servers.

# References

The writing of service subroutines using ATMI functions is the main subject of the *BEA TUXEDO Programmer's Guide*.

Examples of buildserver(1) command lines can also be found in the *BEA TUXEDO Programmer's Guide* and, of course, in Section 1 of the *BEA TUXEDO Reference Manual*.

CHAPTER

# 5 The bankapp Makefile

## A Look at the bankapp Makefile

bankapp includes a makefile that makes all scripts executable, converts data entry masks to binary format, converts the view description file to binary format, and does all the necessary precompiles, compiles and builds to create the application servers. It can also be used to clean up when you want to make a fresh start.

## Editing bankapp.mk

As bankapp.mk is delivered there are a few fields you may want to edit, and some others that may benefit from a little explanation.

### TUXDIR

If you look at bankapp.mk, about 40 lines into the file you come to the following comment and to the TUXDIR parameter:

```
#
# Root directory of TUXEDO System. This file must either be edited to set
# this value correctly, or the correct value must be passed via "make -f
# bankapp.mk TUXDIR=/correct/tuxdir", or the build of bankapp will fail.
#
TUXDIR=../..
```

The TUXDIR parameter should be set to the absolute pathname of the root directory of your BEA TUXEDO system installation.

## APPDIR

You may want to give some thought to the setting of the APPDIR parameter. As bankapp is delivered, APPDIR is set to the directory where the bankapp files are located, relative to TUXDIR. The section in bankapp.mk is as follows:

```
#
# Directory where the bankapp application source and executables live.
# This file must either be edited to set this value correctly, or the
# correct value must be passed via "make -f bankapp.mk
# APPDIR=/correct/appdir", or the build of bankapp will fail.
#
APPDIR=$(TUXDIR)/apps/bankapp
#
```

If you have copied the files to another directory, as is suggested in the README file, you should set this parameter to the name of the directory to which you copied the files. When you run the makefile, the application will be built in this directory.

## NATIVE and Other /Host Parameters

There are some parameters in bankapp.mk that apply to /Host. If you do not have that add-on, you should make sure the parameters are commented out or leave them null.

```
# Directory where the native side source files for CICS host live.
# This file must either be edited to set this value correctly, or the
# correct value must be passed via "make -f bankapp.mk
# NATIVE=/correct/native", or the build of bankapp will fail.
#
NATIVE=$(TUXDIR)/apps/hostapp/cics/native
      .
      .
      .
#
# HOST - set to -DHOST if host credit card processing is desired
#HOST=-DHOST
HOST=
#
```

## Resource Manager

As bankapp is delivered, it expects to use TUXEDO/SQL as the database resource manager. This assumes that you have the BEA TUXEDO system database on your system. If this is not the case, you should set the RM parameter to the name of your resource manager as listed in TUXDIR/udataobj/RM. There is more on this subject in Chapter 6, "Databases for bankapp."

```
#
# Resource Manager
#
RM=TUXEDO/SQL
#
```

# Running bankapp.mk

When you have completed the changes you wish to make to bankapp.mk, run it with the following command line:

```
nohup make -f bankapp.mk &
```

Check the nohup.out file to make sure the process completed successfully.

# 6 Databases for bankapp

## Resource Manager Options for bankapp

This chapter covers the subject of the interface between `bankapp` and a resource manager, typically a database management system. As was mentioned previously, `bankapp` is written to use the BEA TUXEDO/SQL facilities of the BEA TUXEDO system database, which is an XA-compliant resource manager. The first part of the chapter describes how you create the database for `bankapp`.

If you do not have BEA TUXEDO/SQL on your system, you have two options:

- You can integrate an XA-compliant resource manager with the BEA TUXEDO system and bring up `bankapp` with only a few, relatively minor changes.
- You can integrate a non-XA compliant resource manager with `bankapp`, but the required changes are somewhat more extensive.

These two options are discussed in the two later sections of the chapter.

## The System/D RM and bankapp

How you create the `bankapp` database depends on whether you are bringing the application up on a single processor (SHM mode) or on a network of more than one processor (MP mode).

# Create Database in SHM Mode

This is a 2-step procedure.

1. Set the environment by typing the following.

   ```
   . ./bankvar
   ```

   (If you are bringing up `bankapp` in one continuous series of steps, you should have done this earlier. `bankvar` sets a number of parameters that are referenced when `bankapp.mk` is run.)

2. Execute `crbank`. `crbank` calls `crbankdb` three times, changing some environment variables each time, so that you end up with three database files on a single machine. That means you can simulate the multi-machine environment of the BEA TUXEDO system without a network of machines.

# Create the Database in MP Mode

This procedure is quite similar to the one for SHM mode:

1. Set the environment by typing the following.

   ```
   . ./bankvar
   ```

   As noted above, you may already have done this step.

2. Run `crbankdb` to create the database for this site.

3. On each additional machine in your BEA TUXEDO system network, edit `bankvar` to provide the pathname for the `FSCONFIG` variable that is used for that site in the configuration file, `ubbmp`. Then repeat Step 1 and Step 2.

## Failure with a semget Error

If `crbankdb` fails with a `semget` error, it is saying that it cannot get enough semaphores. Each `NPROC` requires two semaphores, but you should be able to reduce the number of processes and still run `bankapp`. Try reducing `NPROCTBL=20` in the `create database` statement in `crbankdb.sh` to `NPROCTBL=10`.

# Using an XA-compliant RM with bankapp

The procedure for integrating an XA-compliant resource manager with the BEA TUXEDO system is provided elsewhere in the BEA TUXEDO documentation; we will not repeat it here. What is described here are changes that need to be made to `bankapp` files to enable you to run with an alternate resource manager.

## Changes to bankvar

The following environment variables are used in creating the BEA TUXEDO system database.

```
BLKSIZE=512
DBNAME=bankdb
DBPRIVATE=no
DIPCKEY=80953
FSCONFIG=${APPDIR}/bankdl1
```

It is unlikely that these correspond to variables needed in creating the database for the alternate resource manager.

## Changes to the bankapp Services

Since all database access in `bankapp` is done with embedded SQL statements, if your new resource manager supports SQL, you should have no trouble. Bear in mind that the utility `appinit.c` includes calls to `tpopen()` and `tpclose()`. `tpopen()` checks the configuration file to learn how to open the application database.

# Change to bankapp.mk

You must edit the `RM` parameter in `bankapp.mk` to name the new resource manager.

Also, the name of the SQL compiler and its options may be different (for example, not `esqlc`). The file suffix may not be `.ec` and the include directory needed to compile the resulting `.c` file may be different.

# Changes to crbank and crbankdb

`crbank` might well be ignored and not used with your alternate resource manager. Its only function is to re-set variables and run `crbankdb` three times. `crbankdb`, on the other hand, requires close attention. In Listing 6-1 we reproduce the beginning of the `crbankdb` script to point out things that won't work with a different resource manager.

**Listing 6-1 An Excerpt from the crbankdb Script**

```
#Copyright (c) BEA Systems, Inc.
#All rights reserved

#
# Create device list
#
dbadmin<<!
echo
crdl
# Replace the following line with your device zero entry
${FSCONFIG}        0         2560
!
#
# Create database files, fields, and secondary indices
#
sql<<!
echo
create database ${DBNAME} with (DEVNAME='${FSCONFIG}',
      IPCKEY=${DIPCKEY},    LOGBLOCKING=0,     MAXDEV=1,
      NBLKTBL=200,          NBLOCKS=2048,      NBUF=70,        NFIELDS=80,
      NFILES=20,            NFLDNAMES=60,      NFREEPART=40,   NLCKTBL=200,
      NLINKS=80,            NPREDS=10,         NPROCTBL=20,    NSKEYS=20,
      NSWAP=50,             NTABLES=20,        NTRANTBL=20,    PERM='0666',
```

```
      STATISTICS='n'
)

create table BRANCH (
      BRANCH_ID             integer not null,
      BALANCE               real,
      LAST_ACCT             integer,
      LAST_TELLER           integer,
      PHONE                 char(14),
      ADDRESS               char(60),
      primary key(BRANCH_ID)
) with (
      FILETYPE='hash',      ICF='PI',      FIELDED='FML',
      BLOCKLEN=${BLKSIZE},  DBLKS=8,       OVBLKS=2
)
!
```

These first forty or so lines will give you an idea of what needs to be changed and what may be salvageable. As you can see, `crbankdb` is made up of two `here` documents that provide input to the `dbadmin` and `sql` shell commands. The first `here` file is passed to the BEA TUXEDO system command `dbadmin` to create a device list for the database. Obviously, this will not work with another resource manager. Other commands may be needed to create table spaces and/or grant the correct privileges.

The second `here` file is passed to System/D's interactive SQL. BEA TUXEDO/SQL conforms closely to the standard SQL, but the `with` clauses of the `create database` and `create table` statements are specific to System/D.

**Note:** In the scripts furnished with `bankapp` the `create table` statement shown in Listing 6-1 is followed by three other `create table` statements and two `create index` statements. The remarks here apply to all of these statements.

# Changes to the Configuration File

This gets a little ahead of our sequence of chapters (configuration files are discussed in Chapter 7, "Edit bankapp Configuration Files."), but you will have to change the `*GROUPS` section to specify a different `TMSNAME` parameter and to provide an `OPENINFO` parameter that is recognizable by the new resource manager.

# Using a non-XA Compliant RM with bankapp

The most significant difference between a resource manager that is not XA-compliant and one that is, is that the non-XA resource manager does not take full advantage of the BEA TUXEDO system Distributed Transaction Processing (DTP) features. Your resource manager will operate as a local resource on the machine on which it resides and clients within a DTP transaction will not be able to request services from your resource manager.

For the discussion at hand, we're going to assume you want to connect an RDBMS that doesn't use the XA 2-phase commit to `bankapp`. The non-XA resource manager will be the only resource manager used by the application; the problem of integrating XA and non-XA resource managers in `bankapp` is not covered in this discussion. You expect to be able to access the database using embedded SQL statements such as those delivered with `bankapp`. The most important change in the functionality of `bankapp` that results from this is that the `TRANSFER` service will no longer be a single, atomic transaction. If a system error should occur between the withdrawal and the deposit in `TRANSFER`, you run the risk of having a corrupted database.

## Changes to bankvar

The following variables can be left null in `bankvar` because they are parameters for the BEA TUXEDO system database.

```
BLKSIZE
DBNAME
DBPRIVATE
DIPCKEY
FSCONFIG
```

The following variable can be left null in `bankvar` because a `TLOG` is needed only for DTP transactions.

```
TLOGDEVICE
```

# Changes to the bankapp Clients and Services

In the `.m` files; that is, the source code for `bankapp` masks, change the following.

```
TRANMODE=TRAN
```

to

```
TRANMODE=NOTRAN
```

In `audit.c` and `auditcon.c` remove the `tpbegin()`, `tpcommit()`, and `tpabort()` statements.

All calls to `tpopen()` and `tpclose()` must be removed. In each service, a local transaction must be started at the beginning of the service and a commit or rollback must be done before each `tpreturn()`. The service `OPEN_ACCT` will need to be re-written since it calls the `DEPOSIT` service, so that the work of `DEPOSIT` is done within the same transaction in the same server. Similarly, `CLOSE_ACCT` calls `WITHDRAW`, and `XFER` calls `DEPOSIT` and `WITHDRAW`. These functions `(DEPOSIT, WITHDRAW)` should be re-written as non-service functions with normal returns that can be called from different service functions.

# Changes to bankapp.mk

In `bankapp.mk`, set RM to null. Change all `buildserver` lines to remove the `-r` flag and to include the libraries needed by your resource manager. A typical `buildserver` line should look like this.

```
buildserver -f servicefile.o -o servername -l "rmlibs,..."
```

The libraries for your resource manager will not be brought in automatically as happens with XA-compliant resource managers that are listed in `TUXDIR/udataobj/RM`, so you have to specify what libraries you need on the `buildserver` command line.

# Changes to crbank and crbankdb

Do not use crbank.

You may be able to salvage some of the create table statements in crbankdb. At any rate, you should plan to use the same table and field names in your database as are used in bankapp in order to be able to use the existing services.

# Changes to the Configuration File

In the *GROUPS section, change the existing entries as follows.

If you are using ubbshm.

```
*GROUPS
DEFAULT:      LMID=SITE1
BANKB1        GRPNO=1
BANKB2        GRPNO=2
BANKB3        GRPNO=3
```

If you are using ubbmp.

```
*GROUPS
DEFAULT:
BANKB1        LMID=SITE1   GRPNO=1
BANKB2        LMID=SITE2    GRPNO=2
```

The above changes do two things: you remove the TMSNAME specification so you default to the null XA interface, and you remove the OPENINFO statements, which are not used with the null XA interface.

In addition to these changes, change the DEFAULT entry for the *SERVICE entries to set AUTOTRAN=N.

# Changes to the Driver Scripts

Edit driver.sh and populate.sh to change the ud -t 30 argument to ud -d 30.

# 7 Edit bankapp Configuration Files

## Configuration Files for bankapp

A configuration file brings together all the detail about how an application maps to the machines on which it runs. As bankapp is delivered, there are two configuration files in the ASCII format described in ubbconfig(5). The file called ubbshm contains the configuration for an application on a single computer. The file called ubbmp contains the configuration for a networked application.

The configuration files are delivered with the value of some parameters enclosed in angle brackets (<>). You need to replace these generic values with values that pertain to your installation. All of these fields occur within the RESOURCES, MACHINES, and GROUPS sections in both files. In ubbmp, the NETWORK section also has entries you must localize. In Listing 7-1 we show ubbmp through the NETWORK section; this illustration also covers all the changes you need to make in RESOURCES, MACHINES, and GROUPS if you are bringing up a single-processor application. An explanation of the values that need to be replaced follows Listing 7-1.

If you want to enable the application password feature, add this line to the RESOURCES section of ubbshm or ubbmp:

```
SECURITY        APP_PW
```

**Listing 7-1 Configuration File Fields to Be Replaced**

```
     #Copyright (c) 1997 BEA Systems, Inc.
     #All rights reserved

     *RESOURCES
     IPCKEY          80952
001  UID             <user id from id(1)>
002  GID             <group id from id(1)>
     PERM            0660
     MAXACCESSERS    40
     MAXSERVERS      35
     MAXSERVICES     75
     MAXCONV         10
     MAXGTT          20
     MASTER          SITE1,SITE2
     SCANUNIT        10
     SANITYSCAN      12
     BBLQUERY        180
     BLOCKTIME       30
     DBBLWAIT        6
     OPTIONS         LAN,MIGRATE
     MODEL           MP
     LDBAL           Y
     #
     *MACHINES
003  <SITE1's uname> LMID=SITE1
004                  TUXDIR="<TUXDIR>"
005                  APPDIR="<APPDIR>"
                     ENVFILE="<APPDIR>/ENVFILE"
                     TLOGDEVICE="<APPDIR>/TLOG"
                     TLOGNAME=TLOG
                     TUXCONFIG="<APPDIR>/tuxconfig"
006                  TYPE="<machine type>"
                     ULOGPFX="<APPDIR>/ULOG"
007  <SITE2's uname> LMID=SITE2
                     TUXDIR="<TUXDIR>"
                     APPDIR="<APPDIR>"
                     ENVFILE="<APPDIR>/ENVFILE"
                     TLOGDEVICE="<APPDIR>/TLOG"
                     TLOGNAME=TLOG
                     TUXCONFIG="<APPDIR>/tuxconfig"
                     TYPE="<machine type>"
                     ULOGPFX="<APPDIR>/ULOG"
     #
     *GROUPS
     DEFAULT: TMSNAME=TMS_SQL   TMSCOUNT=2
     BANKB1   LMID=SITE1         GRPNO=1
```

```
008           OPENINFO="TUXEDO/SQL:<APPDIR>/bankdl1:bankdb:readwrite"
BANKB2        LMID=SITE2        GRPNO=2
009  OPENINFO="TUXEDO/SQL:<APPDIR>/bankdl2:bankdb:readwrite"
*NETWORK
010  SITE1    NADDR="<network address of SITE1>"
011           BRIDGE="<device of provider>"
012           NLSADDR="<network listener address of SITE1>"
013  SITE2    NADDR="<network address of SITE2>"
014           BRIDGE="<device of provider>"
015            NLSADDR="<network listener address of SITE2>"
```

# Notes to Listing 7-1

The following table describes the values you must provide for the angle-bracketed strings.

| Line | Value | Description |
|---|---|---|
| 001 | UID | The effective user ID (UID) for the owner of the bulletin board IPC structures. In a multiprocessor configuration, the value must be the same on all machines. You avoid problems by using the same UID as that of the owner of the BEA TUXEDO system software. |
| 002 | GID | The effective group ID (GID) for the owner of the bulletin board IPC structures. In a multiprocessor configuration, the value must be the same on all machines. Users of the application should share this group ID. |
| 003 | SITE1 name | The name of the machine. Use the value produced by the UNIX command: `uname -n` |
| 004 | TUXDIR | The absolute pathname of the root directory for the BEA TUXEDO system software. Make this a global change to put the value in all occurrences of `<TUXDIR>` in the file. |
| 005 | APPDIR | The absolute pathname of the directory where the application runs. Make this a global change to put the value in all occurrences of `<APPDIR>` in the file. |

| Line | Value | Description |
|---|---|---|
| 006 | machine type | An identifying string. This parameter is important in a networked application where machines of different types are present. The BEA TUXEDO system checks for the value on all communication between machines. Only if the values are different are the message encode/decode routines called to convert the data. |
| 007 | SITE2 name | The name of the second machine. Use the value produced by the UNIX command: uname -n on that machine. |
| 008 | OPENINFO | The statement here and in the following entry are in a format understood by BEA TUXEDO system resource managers. They need to be changed (or removed) to meet the requirements of other resource managers. |
| 009 | Network Address of SITE1 | The full network listening address of the bridge process on this machine. For example addresses, see *Administering the BEA TUXEDO System.* |
| 010 | Device of provider | The full pathname of the device for your network provider. This value should be the same for all entries in the NETWORK section. |
| 011 | Network listener address of SITE1 | The value of the network listener address for the tlisten process on this machine. |
| 012 | Network Address of SITE2 | The full network listening address of the bridge process on this machine. This will be a different value on each machine. |
| 013 | Device of provider | The full pathname of the device for your network provider. This value should be the same for all entries in the NETWORK section. |
| 014 | Network listener address of SITE2 | The value of the network listener address for the tlisten process on this machine. |

# References

All of the configuration parameters and their values are described in ubbconfig(5) in the *BEA TUXEDO Reference Manual*.

As noted above, there are examples of the proper format for network address parameters in *Administering the BEA TUXEDO System*.

# 8 Create tuxconfig, tlog; Start tlisten

## Creating tuxconfig, tlog tlisten

This chapter describes how to prepare to boot bankapp.

You will find that most of the material applies to a networked application, that is, a configuration with more than one machine. If you are bringing bankapp up in SHM mode, you do not have to be concerned about the tlisten process or about creating a TLOG on another machine.

As with all the steps since Chapter 2, “bankapp Files,” of this guide, you should be in the directory in which your bankapp files are located and you must set the environment by entering.

```
. ./bankvar
```

## Loading the Configuration File

Once the configuration file has been edited to your satisfaction, it must be loaded to a binary file on your MASTER node. The binary configuration file has a file name of tuxconfig; its pathname relative to APPDIR is in the environment variable, TUXCONFIG. The file should be created by a person with the effective user ID and group ID of the BEA TUXEDO system administrator, which should be the same as the

UID and GID values in your configuration file. If these conditions are not observed, you may run into permission problems in running bankapp. The command line for creating tuxconfig is:

```
tmloadcf ubbmp
```

There is a -y option to suppress prompts that ask if you really want to install TUXCONFIG or to overwrite it if it already exists. There is a -c option that calculates the numbers for IPC resources the configuration requires.

tuxconfig needs to be installed only on the MASTER node; it is propagated to other nodes by tmboot when the application is booted.

If you have specified SECURITY as an option for the configuration, tmloadcf prompts you to enter an application password. The password you select can be up to 30 characters long. Client processes joining the application will be required to supply the password.

tmloadcf parses the ASCII configuration file for syntax errors before it loads it, so if there are errors in the file, the job fails.

# Creating the TLOG

The TLOG is the transaction log needed by the BEA TUXEDO system in the management of global transactions. Before an application can be booted an entry for the TLOG must be created on all nodes of the application, and a file for the log itself must be created on the MASTER node.

**Note:** In a production environment, the device list may be the same as that used for the database. (See *Administering the BEA TUXEDO System*.)

There is a script in bankapp called crtlog that creates the device list and the TLOG for you. The device list is created using the TLOGDEVICE variable from bankvar. On the MASTER node, enter the command as follows.

```
crtlog -m
```

On all other machines, do not specify -m; when the system is booted, the BBL on each non-MASTER node creates the log.

If you are using a non-XA resource manager, there is no requirement for a transaction log so you may skip this step.

# Starting tlisten

`tlisten` is the ProductName listener process that provides a remote service connection between nodes of an application for ProductName processes such as `tmboot`. It must be installed on all nodes of your network as defined in the `NETWORK` page of the configuration file.

Starting `tlisten` is described in more detail in the *BEA TUXEDO Installation Guide,* as a step in the installation of the ProductName software. For the purposes of running `bankapp` you may prefer to start a separate instance. It can be done with a command like this.

```
tlisten -d /dev/devname -l nlsaddr
```

where *`devname`* is the device name of your network provider. This is apt to be `/dev/tcp`. (If your provider is `Sockets`, the `-d` option is not needed.)

The *`logfile`* used by `tlisten` is separate from all other BEA TUXEDO system log files, but one log can be used by more than one `tlisten` process. The default filename is `$TUXDIR/udataobj/tlog`.

The *`nlsaddr`* value must be the same as that specified for the `NLSADDR` parameter for this machine in your configuration file. As noted in the previous chapter, this value changes from one machine to another; it is important that your `tlisten` arguments agree with your configuration file specification.

**Note:** Detection of an error in this specification is not easy. `tmloadcf` does not check for agreement between your configuration file and your `tlisten` command. The symptom is most likely to be that the application fails to boot on the machine where the mismatch in *`nlsaddr`* values occurs or where the `tlisten` process has not been started.

## Stopping tlisten

`tlisten` is designed to run as a *daemon* process. The reference page has some suggestions about incorporating it in startup scripts or running it as a `cron` job. For `bankapp`, you may prefer simply to start it and bring it down as you need it. To bring it down, send it a `SIGTERM` signal like this.

```
kill -15 pid
```

## Error Messages from tlisten Problems

If no remote `tlisten` is running, the boot sequence is displayed on your screen as follows.

```
Booting admin processes…

exec DBBL -A :
      on MASTER -> process id=17160…Started.
exec BBL -A :
      on MASTER -> process id=17161…Started.
exec BBL -A :
      on NONMAST2 -> CMDTUX_CAT:814: cannot propagate TUXCONFIG file

tmboot: WARNING: No BBL available on site NONMAST2.
      Will not attempt to boot server processes on that site.

exec BBL -A :
      on NONMAST1 -> CMDTUX_CAT:814: cannot propagate TUXCONFIG file

tmboot: WARNING: No BBL available on site NONMAST1.
      Will not attempt to boot server processes on that site.

2 processes started.
```

and messages such as these will be in the ULOG:

```
133757.mach1!DBBL.17160: LIBTUX_CAT:262: std main starting
133800.mach1!BBL.17161: LIBTUX_CAT:262: std main starting
133804.mach1!BRIDGE.17162: LIBTUX_CAT:262: std main starting
133805.mach1!tmboot.17159: LIBTUX_CAT:278: Could not contact NLS on NONMAST2
133805.mach1!tmboot.17159: LIBTUX_CAT:276: No NLS available for remote
  machine NONMAST2
133806.mach1!tmboot.17159: LIBTUX_CAT:276: No NLS available for remote
  machine NONMAST2
133806.mach1!tmboot.17159: CMDTUX_CAT:850: Error sending TUXCONFIG
  propagation request to TAGENT on NONMAST2
133806.mach1!tmboot.17159: WARNING: No BBL available on site NONMAST2.
       Will not attempt to boot server processes on that site.
133806.mach1!tmboot.17159: LIBTUX_CAT:278: Could not contact NLS on NONMAST1
133806.mach1!tmboot.17159: LIBTUX_CAT:276: No NLS available for
  remote machine NONMAST1
133806.mach1!tmboot.17159: LIBTUX_CAT:276: No NLS available for
  remote machine NONMAST1
133806.mach1!tmboot.17159: CMDTUX_CAT:850: Error sending TUXCONFIG
  propagation request to TAGENT on NONMAST1
133806.mach1!tmboot.17159: WARNING: No BBL available on site NONMAST1.
        Will not attempt to boot server processes on that site.
```

If `tlisten` is started with the wrong machine address, the following messages appear in the `tlisten` log.

```
Mon Aug 26 10:51:56 1991; 14240; BEA TUXEDO System Listener Process Started
Mon Aug 26 10:51:56 1991; 14240; Could not establish listening endpoint
Mon Aug 26 10:51:56 1991; 14240; Terminating listener process, SIGTERM
```

# References

For more information about `tlisten` and the `TLOG`, see Chapter 15, “Monitoring Log Files,” in *Administering the BEA TUXEDO System*.

For examples of network addresses, see Chapter 6, “Building Networked Applications,” in *Administering the BEA TUXEDO System*.

Installation of `tlisten` is covered, as noted above, in the *BEA TUXEDO Installation Guide*.

The following pages in the *BEA TUXEDO Reference Manual* are important.

- `tlisten`(1)
- `tmadmin`(1) for the `crdl` and `crlog` commands
- `tmloadcf`(1)

*CHAPTER*

# 9 Boot the Application; Populate the Database

## tmboot and populate

This chapter covers booting the application and putting enough records into the database to simulate a real application.

## Checking IPC Resources

When your application is defined to the point where you are ready to boot it, you should first run a check to make sure your machine has enough IPC resources to support your application. The `tmboot` command has a `-c` option that produces a report like that shown in Listing 9-1.

**Listing 9-1 tmboot -c IPC Report**

```
Ipc sizing (minimum /T values only)…

                Fixed Minimums Per Processor

SHMMIN: 1
SHMALL: 1
SEMMAP: SEMMNI

                Variable Minimums Per Processor

               SEMUME,          A                               SHMMAX
                SEMMNU,          *                                   *
Node           SEMMNS  SEMMSL  SEMMSL  SEMMNI  MSGMNI  MSGMAP  SHMSEG
------         ------  ------  ------  ------  ------  ------  ------
sfpup              60       1      60   A + 1      10      20     76K
sfsup              63       5      63   A + 1      11      22     76K

where 1 <= A <= 8.
```

The number of expected application clients per processor should be added to each `MSGMNI` value. `MSGMAP` should be twice `MSGMNI`.

The minimum IPC requirements can be compared to the parameters set for your machine. The most likely place to find the settings on a UNIX system machine is in the file `/etc/conf/cf.d/mtune`, but this can vary from one platform to another and between versions of the UNIX operating system. See the system administrator's guide for your machine for information about how to find and change these parameters. If you are using the BEA TUXEDO system on a Windows NT platform, there is a control panel that displays and sets IPC parameters.

# Executing tmboot

As with most procedures in this guide, we start by setting the environment:

```
../bankvar
```

The variables particularly needed by `tmboot` are `TUXCONFIG`, `APPDIR`, and `TUXDIR`. The command to boot the complete application is the following.

```
tmboot
```

Running this command causes the following prompt to be displayed.

```
Boot all admin and server processes? (y/n): y
```

When you respond `y` to the prompt, you get a running report that starts like this.

```
Booting all admin and server processes in /usr/me/appdir/tuxconfig
Booting all admin processes…
exec BBL -A:
       process id=24223… Started.
```

The display continues until all servers in the configuration have been started. It ends with a count of the number of servers started.

There are options that can be used to boot only a portion of the configuration. For example, if the `-A` flag is used, only administrative servers are booted, but with no options specified, everything is booted.

In addition to the report on servers booted, `tmboot` also sends messages to the `ULOG`.

## The Userlog: ULOG

We have referred previously to the `ULOG`, but this is the first time it has actually played an important role in the process under discussion. It is called `ULOG` (short for user log) because that is the default prefix; the actual file name of the log is `ULOG` followed by the date in the form: `.`*mmddyy*. Log messages can be directed to `ULOG` from user-written modules through a call to `userlog`(3c), but the `ULOG` is also used heavily by BEA TUXEDO system processes such as `tmboot`.

# Running the populate Script

The populate.sh script is provided with bankapp to put enough records into the database to work with. populate is a one-line script that pipes records from a program called gendata to the system server, ud. The gendata program creates records for 10 branches, 30 tellers, and 200 accounts. A file of the records created is kept in pop.out, so you can use values that are in the database when forming your sample service requests. The script is run just by entering the following word.

```
populate
```

# References

For more information about tmboot, see Chapter 4, “Starting and Shutting Down Applications,” in *Administering the BEA TUXEDO System*.

Chapter 7, “Error Management,” of the *BEA TUXEDO Programmer's Guide* contains background information on the user of the userlog. Throughout that guide there are examples of messages being sent to the log.

The following pages in the *BEA TUXEDO Reference Manual* are important:

- tmboot(1)
- ud(1)
- userlog(3c)

# 10 Run bankapp

## Run the Application

This chapter covers some of the scripts and commands you can use after bankapp has been booted.

We recognize the probability, since you have a system that is active, that you already have set the bankapp environment. However, if that is not the case, if you are logging in cold to a running system, you will need to enter the following.

```
. ./bankvar
```

to set your environment for bankapp.

## The bankapp run Script

A script called run is provided with bankapp. This script brings up the initial menu with its choice of six services you can request bankapp to perform. run contains a single command line:

```
mio -i MENU
```

where the -i option tells mio to use the MENU mask rather than the default, which prompts for the name of the mask to use.

You might want to enter the mio command directly, just to see what happens. There is a HELP screen that gives you a summary of a number of keystrokes that enable you to move around in mio masks.

The output file that was created by the populate script, pop.out, can be used to provide account numbers, branch IDs, and other fields you can specify on the data entry masks, so your service requests produce some output.

# Running the audit Client Program

The `audit.c` client program was described in Chapter 3, "bankapp Client Programs." To execute the program, enter the command line as follows.

```
audit {-a | -t} [branch_id]
```

specifying either `-a` for account balances or `-t` for teller balances. If you specify a *branch_id*, the report is limited to that branch; if you do not specify a *branch_id*, the report is for all branches.

# Running auditcon

To start the conversational version of the `audit` program, enter the command.

```
auditcon
```

The program displays the following message on your terminal.

```
to request a TELLER or ACCOUNT balance for a branch,
type the letter t or a, followed by the branch id,
followed by <return>

for ALL TELLER or ACCOUNT balances, type t or a <return>
q <return> quits the program
```

When you have typed your request and pressed return, the requested information is displayed on your terminal followed by this.

```
another balance request ??
```

The program continues to offer you this service until you enter a `q`.

# Using the driver Program

The `driver` program is a script that generates a series of transactions to simulate activity on the system. It is included as part of the sample application so you can get realistic-looking statistics with commands of the `tmadmin` interface. By default, the `driver` program generates 300 transactions. You can change that number with the `-n` option, as in the following example.

```
driver -n1000
```

specifies that the program should run for 1,000 loops.

# Using tmadmin

This book is not the place to go into an extensive description of the BEA TUXEDO system administrative interface, `tmadmin`. We encourage you to use it while `bankapp` is running to see the kind of information you can produce with `tmadmin` subcommands.

# Shutting Down bankapp

When you want to bring `bankapp` down, enter the `tmshutdown`(1) command with no arguments, as follows.

```
tmshutdown
```

Running this command (or the `shutdown` command of `tmadmin`) will cause the shutting down of all application servers, gateway servers, TMSs, and administrative servers, and the removal of associated IPC resources.

The `shutdown` command must be issued from the `MASTER` machine.

# References

For more information about using `tmadmin`, the command-line interface for administration, see Chapter 14, "Monitoring a Running System," in *Administering the BEA TUXEDO System*.

The following pages of the *BEA TUXEDO Reference Manual* are important:

- `mio`(1)
- `tmadmin`(1)
- `tmshutdown`(1)