BEA TUXEDO

Workstation Guide

BEA TUXEDO Release 6.5
Document Edition 6.5
February 1999

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA TUXEDO Workstation Guide

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

Contents

1. Overview of BEA TUXEDO Workstation

2. BEA TUXEDO System Workstation Administration

3. BEA TUXEDO Workstation for UNIX System Workstations

4. BEA TUXEDO Workstation for MS-DOS Workstations

5. BEA TUXEDO Workstation for WINDOWS

6. BEA TUXEDO Workstation for OS/2

7. BEA TUXEDO Workstation for Macintosh

CHAPTER

1 Overview of BEA TUXEDO Workstation

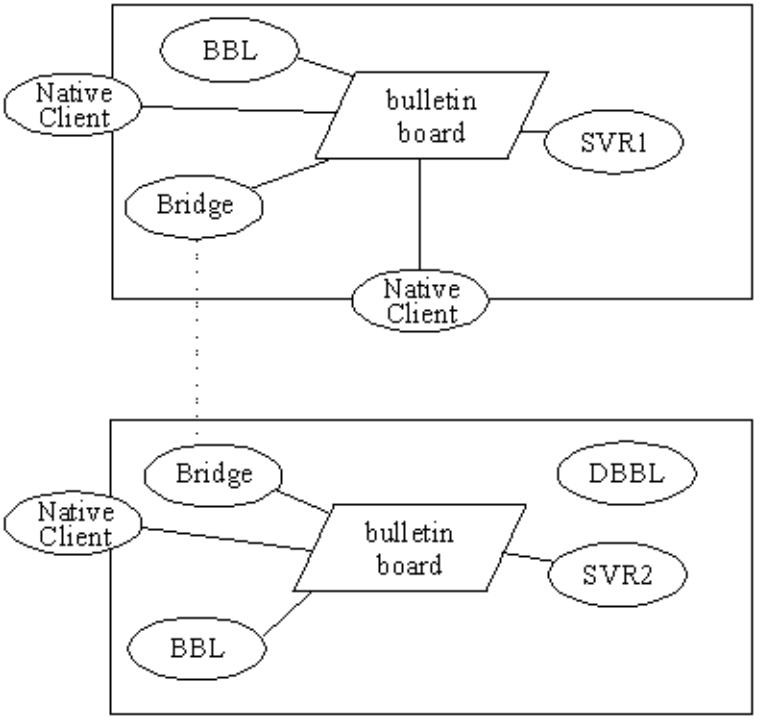
Overview of the BEA TUXEDO Workstation Product

This chapter lists the features of the BEA TUXEDO Workstation product. It also shows how Workstation alters the existing boundaries of the BEA TUXEDO system, its added features, and the location of Workstation components.

Product Perspective of BEA TUXEDO Workstation

In prior releases, BEA TUXEDO system applications were capable of distributing services across networks of UNIX-based processors. However, all the processors, servers, services, and transactions of the application, and the bridge processes that connect the nodes had to be defined in a configuration file used by the BEA TUXEDO boot process to start the application. Such an application might be diagrammed as shown in Figure 1-1.

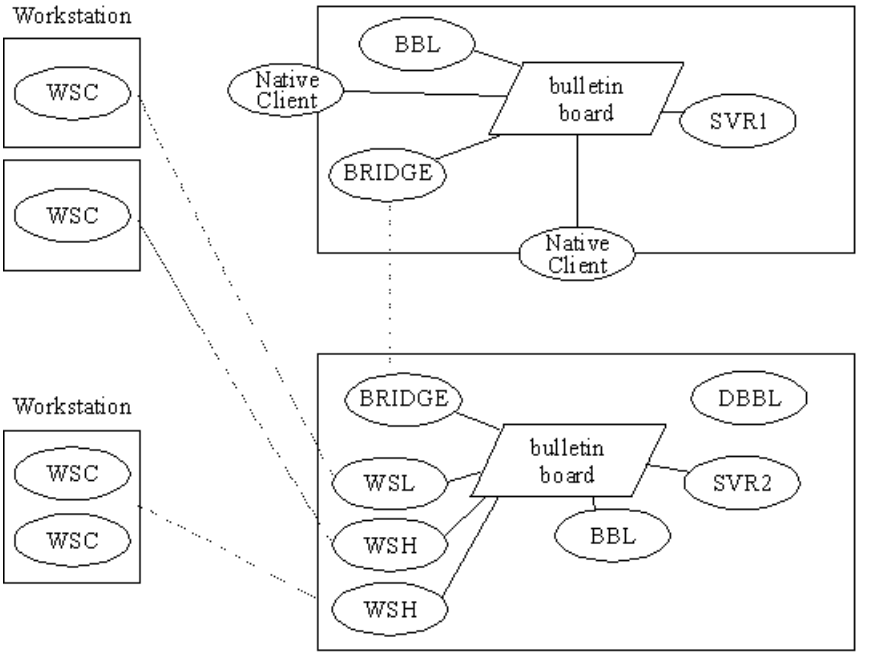
Figure 1-1 BEA TUXEDO Application without the Workstation Feature



Client processes (shown as Native Clients in Figure 1-1) are invoked by users logged in directly on a node where a bulletin board for their application exists under the control of a BBL. They are considered to be within the administrative domain of the application.

Workstation extends the availability of a native BEA TUXEDO application, like the one shown above, to clients resident on workstations. With the Workstation feature, workstations are not required to be within the administrative domain of the application. The Workstation instead, defines an environment where clients can access services of an application through a surrogate handler process. A BEA TUXEDO application using Workstation might be diagrammed as shown in Figure 1-2.

Figure 1-2 BEA TUXEDO Application with the Workstation Feature



WSC: Workstation Client

WSL: Workstation Listener

WSH: Workstation Handler

On the workstation, the programming environment is one that works under the operating system of the machine. A Local Area Network provides connectivity to the administrative domain of the application. Therefore, workstations can be UNIX-based as well as non-UNIX based (MS Windows, for example). This gives greater flexibility in the choices of hardware and software platforms on which to deliver the services of your BEA TUXEDO system applications.

Features of BEA TUXEDO Workstation

The function of Workstation is to provide access to BEA TUXEDO system applications from a network of workstations. The BEA TUXEDO system accomplishes this by providing the same application programming interface (API) for Workstation clients that is available for native-site clients.

Note: Existing native-site client programs can usually be moved to Workstation sites without modification beyond a recompile.

A potential benefit of Workstation is an increase in an application's performance. Before the Workstation feature, the native processor executed all the client code as well as the server and BEA TUXEDO code. With Workstation, the CPU cycles formerly needed by clients are now available to servers and the BEA TUXEDO system. The client CPU cycles are now furnished by the Workstation processors. The following features are being introduced as part of Workstation:

- Customization of the workstation handler so applications can define new buffer types
- A client development environment that includes compilation tools so clients can be compiled and link edited for specific types of workstations

In addition, Workstation clients can be written to take advantage of other BEA TUXEDO system features such as client naming. Client naming enables:

- Unsolicited notification—clients can receive out-of-band messages from other clients or from servers.
- Authentication—password protection for an application or client-by-client authentication can be implemented.
- Client statistics—`tmadmin`(1) can monitor statistics of the work load and status of clients.

These features are described in the *BEA TUXEDO Programmer's Guide* and *Administering the BEA M3 System*, as well as in Section 1 of the *BEA TUXEDO Reference Manual*.

What Goes Where?

Formerly, all of the BEA TUXEDO system software was installed on all of the machines of an application. Now we need to distinguish between software for the BEA TUXEDO system administrative domain, software for the development environment, and platform-specific software needed only at runtime for specific Workstation clients.

For the Administrative Domain

The complete release must be installed on the principal server machine. Newer BEA TUXEDO system releases include changes to existing commands and libraries to support the client naming and authentication features. Also included in the recent releases are the workstation listener and workstation handler. The listener and handler must be installed on the machine running the highest level BEA TUXEDO release.

The Workstation can be installed as a step in the installation of the complete BEA TUXEDO system or it can be added at a later time. If you install everything at one time, consult the procedures in the *BEA TUXEDO Installation Guide*. If you install it separately, refer to the installation instructions in the box containing your software.

For the Workstation Client Development Environment

The exact nature of the Workstation client development software differs according to the platform for which it is intended, but in all cases the directory structure (under `$TUXDIR`, the root BEA TUXEDO system directory) is as follows.

`include`
: Application-visible `include` files such as `atmi.h`.

`bin`
: BEA TUXEDO system commands, for example, `buildclt`(1).

`locale/C`
: Default message catalog directory (English version).

`lib`
: Libraries necessary for building clients.

The location of this software also depends on the characteristics of the machines on which your BEA TUXEDO system application runs. In general, you will probably choose to do client development work on a limited number of machines of the type you expect to use for workstations. Executable client programs can then be moved to the machines where they will be used.

For the Workstations

To use BEA TUXEDO Workstation, install the package from the CD-ROM. Consult Appendix A of the *BEA TUXEDO Installation Guide* for specific information about installing BEA TUXEDO on your platform(s).

2 BEA TUXEDO System Workstation Administration

What This Chapter Is About

This chapter describes the administration issues for BEA TUXEDO system applications that use the Workstation feature. The administration of Workstation clients, Workstation handlers, and the listener are all performed in the BEA TUXEDO system administrative domain, not at the workstation. The issues discussed here are Workstation specific; they do not include topics such as client naming and authentication that apply to native clients as well as Workstation clients. For a complete discussion of BEA TUXEDO system application administration, please refer to *Administering the BEA TUXEDO System*.

Issues covered here include:

- Configuring BEA TUXEDO Workstation
- Workstation Client Timeout
  - The Keep-alive Option
  - The Network Timeout Option

Configuring BEA TUXEDO Workstation

The UBBCONFIG file contains important Workstation parameters in the RESOURCES and MACHINES sections. The Workstation listener is defined as a system-supplied server in the SERVERS section, with a reference to an entry in the GROUPS section. We will examine the pertinent sections in the order in which they appear in the configuration file.

RESOURCES Section and MACHINES Section

The MAXWSCLIENTS parameter can be specified in the MACHINES section of the configuration file to apply to specific machines. MAXWSCLIENTS is the only parameter that has special significance for Workstation. If MAXWSCLIENTS is not specified, the default is 0.

MAXWSCLIENTS

The MAXWSCLIENTS parameter tells the BEA TUXEDO system, at boot time, how many *accesser slots* to reserve exclusively for workstation clients. For native clients, each accesser slot requires one semaphore. However, the Workstation handler process (executing on the native platform on behalf of workstation clients) multiplexes Workstation client accesses through a single accesser slot, and therefore requires only one semaphore. This points out an additional benefit of the Workstation extension. By putting more clients out on workstations and off the native platform, an application reduces its IPC resource requirements. To repeat, although MAXWSCLIENTS is optional, if not specified, the default is 0.

MAXWSCLIENTS takes its specified number of accesser slots from the total set in MAXACCESSERS. This is important to remember when specifying MAXWSCLIENTS; enough slots must be left to accommodate native clients as well as servers. If you specify a value for MAXWSCLIENTS greater than MAXACCESSERS, native clients and servers will fail at tpinit() time.

GROUPS Section

A GROUPS section entry is required for the group that includes the Workstation listener; the listener need not be the only member of the group. The *GROUPNAME* is a name selected by the application. The following two parameters must be specified for each entry:

LMID = *identifier* [,*identifier*]
: The *identifier(s)* given as the value of the LMID parameter must be among those specified in the MACHINES section.

GRPNO = *number*
: The value is a number between 1 and 30,000 and must be unique among all entries in the GROUPS section.

No other parameters of the GROUPS section need be specified for the group that includes the Workstation listener; in fact, none of the other parameters make any sense for the listener. If you choose, you can specify other servers in the group, but we recommend that you do not include any that expect to open a resource manager.

SERVERS Section

Workstation clients access your application through the services of:

- A Workstation listener process
- One or more Workstation handler processes

The Workstation listener (WSL) and Workstation handler (WSH) are specified in one entry as a BEA TUXEDO system-supplied server, although they are separate processes. The WSL can support multiple Workstation clients. It acts as the single point of contact for all the Workstation clients that connect to your application at the network address specified on the WSL command line. The listener schedules work for one or more Workstation handler processes. A Workstation handler process acts as a surrogate within the administrative domain of your application for clients on remote workstations. The WSH uses a multiplexing scheme to support multiple Workstation clients concurrently.

The tag, or *AOUT*, value for a Workstation listener entry in the SERVERS section is WSL.

Each entry must have the following parameters:

`SRVGRP = groupname`
: this will be the previously defined *GROUPNAME* value from the `GROUPS` section.

`SRVID = number`
: is a number between 1 and 30,000 that identifies a server within its group.

`CLOPT = options`
: are the command line options for the `WSL`. They are described in the section that follows and also on the `WSL`(5) reference page in the *BEA TUXEDO Reference Manual*.

Other `SERVERS` section parameters that are useful for `WSL` entries are:

`SEQUENCE = number`
: gives control over the order in which servers are booted (also used by `tmshutdown`).

`RESTART = {Y|N}`
: should be specified as `Y` to permit restarts.

`GRACE = number`
: should be specified as `0` to permit an infinite number of restarts.

Other optional `SERVERS` parameters are described in `ubbconfig`(5).

Workstation Client Timeout

In earlier releases (prior to v6.4), the loss of a network connection (because of a problem in the network, the `WSH`, or the server) caused another problem from which it was hard to recover: the application would hang indefinitely, while the client waited for a response from the `WSH`. This problem was not limited to API calls that received data, such as `tpgetreply`(3c) or `tprecv`(3c). It occurred in all API calls to the `WSH` except `tpbegin`(3c), and in function calls for buffer allocation.

The current release provides two administrative options to `WSL` that enable you to avoid this problem. Specifically, these options allow you to:

- Check client connections periodically (the keep-alive option)

- Limit the amount of time that a client waits for a response from a WSH before dropping the connection to that WSH (the network timeout option)

With these features, you can ensure that the WSH client no longer hangs indefinitely when a network connection is lost.

This section describes these features and provides instructions for using them.

The Keep-alive Option

Keep-alive is a networking operation that periodically checks the viability of a network connection between WSH and a workstation client if no traffic has occurred over that connection within a period of time specified by the operating system.

How Keep-alive Works

You can request the keep-alive operation through a new administrative option to the WSL: -K. This option improves the speed and reliability of detection of a network failure, by actively testing the state of an idle connection at the protocol stack level. The -K option can be set to client, handler, both, or none.

- If the -K client option is used, keep-alive messages are generated from the client machines. If the keep-alive message is not acknowledged, the network is considered down by the client machine. Subsequent ATMI calls fail with a tperrno of TPESYSTEM.
- If the -K handler option is used, keep-alive messages are generated from the handler machine. If the keep-alive message is not acknowledged, the network is considered down by the handler machine. The handler can then clean up the entry associated with the client that does not respond. This reduces the possibility that the handler will exhaust its mpx factor (as specified by -x) with stale clients.
- If the -K both option is used, keep-alive messages are generated from both the client and handler machines.The availability and timeout thresholds for this feature are determined by tunable parameters in the operating system.
- The none option can be specified explicitly. Doing so has the same effect as not specifying -K at all.

Limitations

The keep-alive operation is supported only on platforms on which the BEA TUXEDO system uses sockets:

- AIX
- AS/400
- Digital UNIX
- HP/UX
- MVS/OE
- Sequent
- Windows

You cannot use this option on any other platform. The BEA TUXEDO system lets you specify the `-K` option for any server machine, but it will not execute it properly on any platform other than those previously listed. If you try to perform a keep-alive operation on any other platform, your attempt fails and a message is written to the userlog (once per process for the `WSH`). Processing continues normally:

- The `-K` option is ignored by pre-6.4 clients.
- The keep-alive operation works only for TCP/IP communications.
- `WSL` command-line options cannot be updated through the `WSL` form in the WEB GUI, but they can be updated through the `CLOPT` option on the server form of the GUI.

How to Use Keep-alive

To use the keep-alive operation in your BEA TUXEDO application, edit the `UBBCONFIG` file as follows.

1. Open your `UBBCONFIG` file.
2. Go to the `SERVERS` section.
3. Find your entry for `WSL`.

4. To the CLOPT argument, add the -K option, followed by one of four valid arguments: client, handler, both, or none (the default).

To Activate Keep-alive for	Select
The Workstation client	client
The Workstation Handler	handler
Both the client and the handler	both
Neither the client nor the handler	none (the default)

Your entry in the UBBCONFIG file should look like the following.

```
WSL SRVGRP="WSLGRP" SRVID=1000 RESTART=Y GRACE=0
CLOPT="-A -- -n 0x0002ffffaaaaaaaa -d /dev/tcp -K both"
```

In the example, -K turns on the KEEPALIVE checking on both the Workstation client and the server side.

For details about the format of a WSL entry in UBBCONFIG, see WSL(5) in the *BEA TUXEDO Reference Manual*.

Note: Any timeout period that you specify applies to the entire system. Remember that if, with one application in mind, you later change the amount of time specified, any other application that uses keep-alive is also affected.

The Network Timeout Option

Network timeout is an option that lets you decide how long you are willing to wait for an operation in a Workstation client before your request for that operation is cancelled (times out) on a network.

You can request the network timeout function through an administrative option to the WSL: -N. The -N option uses a network timeout to receive data in the Workstation client.

How Network Timeout Works

The network timeout option establishes a waiting period (in seconds) for any BEA TUXEDO operation in the Workstation client that receives data from the network. If the period is exceeded, the operation fails and the client is disconnected from the application. A value of 0 (the default) indicates no timeout.

Note: Setting this value too low may cause too many disconnects.

Each ATMI call returns an error whenever a timeout occurs. When a link times out, the application is notified. An existing error code is used. (Additional error detail on the specific error can be retrieved by a call to `tperrordetail`(3c)). Once a network timeout occurs, the status of outstanding operations is in doubt: transactions cannot be completed; incoming replies can be lost, and so on. The only safe action is to terminate the connection to the application by doing the equivalent of a `tpterm`(3c) without communicating with the `WSH`.

By the time the operation returns, the client is no longer part of the BEA TUXEDO application. The application can rejoin the application in either of two ways:

- By calling `tpinit`(3c)
- By using an implicit connection (if security is not configured)

Limitations

- The `-N` option for Workstation client network timeout is supported on all platforms except Mac. Mac Workstation clients are disconnected if this feature is enabled. To support both non-Mac clients with this option and Mac clients without this option, it is necessary to configure more than one `WSL`.
- The `-N` option is ignored by pre-6.4 clients.
- Network timeout does not handle network send operations.
- Timeout periods override any existing blocking or transaction timeout that is in effect.
- Network timeout disconnects the Workstation client after timeout even though the connection may still be viable.

- WSL command-line options cannot be updated through the `WSL` form in the WEB GUI, but they can be updated through the `CLOPT` option on the server form of the GUI.

How to Use Network Timeout

To use the network timeout function in your BEA TUXEDO application, edit the `UBBCONFIG` file as follows.

1. Open your `UBBCONFIG` file.
2. Go to the `SERVERS` section.
3. Find your entry for `WSL`.
4. To the `WSL CLOPT` argument, add the `-N` option.

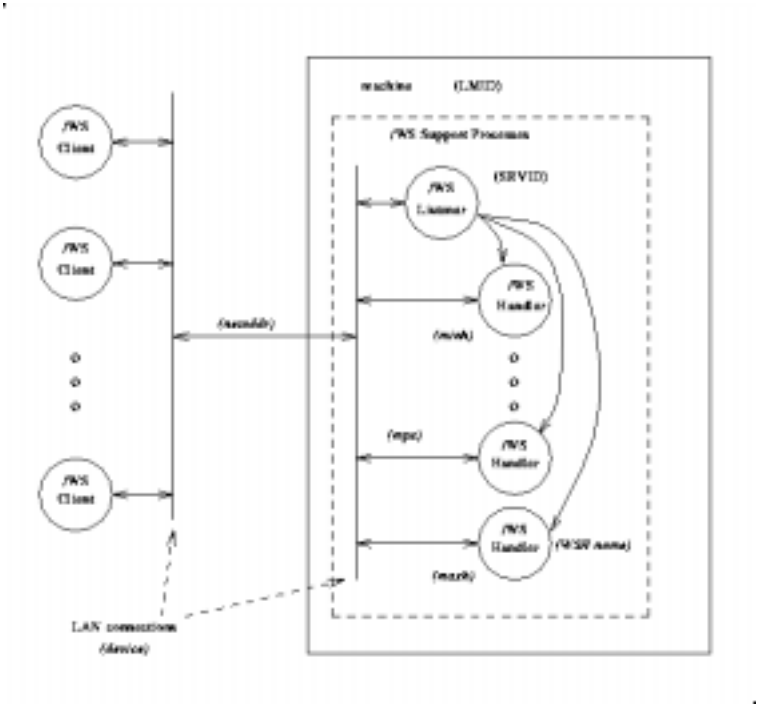
The WSL CLOPT Parameter

The command-line options specified via the `CLOPT` parameter tell the system:

- What services the server will offer. This must be specified as `-A`.
- The network address where Workstation clients connect to the listener
- What network device Workstation listener uses (for example, `/dev/tcp` or `/dev/starlan`).
- The name of the Workstation handler process
- How many Workstation handlers to start (minimum and maximum)
- How many Workstation clients are to be multiplexed through each Workstation handler
- Timeout factor for slow network delays
- Keep-alive option for Workstation clients or `WSH`
- Timeout for inactive client processes

Figure 2-1 illustrates the relationship between Workstation clients and the support processes on the machine serving as the administrative domain. The words in parentheses are substitutables in the command line options of the WSL or server entry parameters.

Figure 2-1 BEA TUXEDO Workstation Feature: Support Processes



The format of the CLOPT parameter is as follows:

```
CLOPT="[-A] [servopts options] - - -n netaddr [-d device] \
   [-w WSHname] [-t timeout-factor] [-T client-timeout] \
   [-m minh] [-M maxh] [-x mpx-factor] \
   [-p minwshport] [-P maxwshport] \
   [-I init-timeout] [-c compression-threshold] [-k compression-threshold] \
   [-z bits] [-Z bits] [-H external_netaddr] [-K {client|handler|both|none}]"
```

As noted above, the `-A` value indicates that the `WSL` is to be booted offering all its services. This is a default, but it is shown in the example to emphasize the distinction between system-supplied servers and application servers. The latter may be booted offering only a subset of their available services.

The `--` syntax marks the beginning of parameters that are passed to the `WSL` after it has been booted.

The -n netaddr Option

This is required. *`netaddr`* is the network address used by Workstation clients to connect to your application. The Workstation listener process uses this address to listen for clients attempting to connect at this address.

The -d device Option

The value of *`device`* is the device name used by the Workstation listener to access the network. In the past this option was required if the transport provider was TCP/IP. As of BEA TUXEDO Release 6.4, this option is no longer required.

The -w WSHname Option

`WSHname` is the name of the Workstation handler process started by the Workstation listener. You should specify the name of the executable created by `buildwsh`(1). The default name assumed for that `a.out` is `WSH`, the Workstation handler that is delivered with Workstation. That name is used if the `-w` option is omitted. If you customize a Workstation handler, make certain that it resides in either `$APPDIR` or `$TUXDIR/bin`. If your customized Workstation handler is used by only one application, we recommend putting the handler in the directory specified by the `APPDIR` environment variable. If the handler is used by many applications, we recommend putting it in the `bin` directory under `$TUXDIR`.

The -t timeout-factor Option

When a large number of workstation clients attempt to connect simultaneously, some of the requests may time out due to a blocking condition. The `-t` *`timeout-factor`* is provided to guard against this possibility. The value of *`timeout-factor`* is a number that, when multiplied by `SCANUNIT`, specifies the amount of time in seconds that

should be allowed for a workstation client to complete initialization processing through the WSH before being timed out by the WSL. The default for this parameter is 3 in a non-security application, and 6 in a security application.

The -T client-timeout Option

Client-timeout is the amount of time (in minutes) that a client is allowed to stay idle. If a client does not make any requests within this time period, the WSH disconnects the client. If this argument is not given or is set to 0, then the timeout is infinite.

The -m minh Option

The *minh* value is the minimum number of Workstation handler processes to be started by this Workstation listener when the listener is booted. If specified, the -m option takes a value from 0 to 255. The default is 0.

The -M maxh Option

maxh is the maximum number of Workstation handler processes to be started by this Workstation listener. If specified, the -M option takes a value from 1 to 32,767. The default is MAXWSCLIENTS for the machine divided by the multiplexing factor MPX. For example, let's say you specify MAXWSCLIENTS=100 in the MACHINES section and -x10 for the CLOPT of the listener. The default for -M *maxh* is 100/10 or 10. *maxh* must be greater than or equal to *minh*.

The -x mpx-factor Option

The *mpx-factor* value specifies the number of Workstation clients you want each Workstation handler to support simultaneously. If specified, the -x option takes a value from 1 to 32,767. The default is 10.

The -K { client | handler | both | none } Option

Use -K to turn on the network keep-alive operation for the client, the handler, or both. You can turn off this option for both the client and handler by specifying none.

Example

The entries related to Workstation as specified in the configuration file for the bankapp application are shown in Listing 2-1.

Listing 2-1 Excerpts from bankapp configuration file

```
#
# ubbconfig file for Workstation example, SHM mode
#
"mach1"         LMID="SITE1"
                TUXCONFIG="/tuxroot/tuxapp/tuxconfig"
                TUXDIR="/tuxroot"
                APPDIR="/tuxroot/tuxapp"
                TLOGDEVICE="/tuxroot/tuxapp/TLOG"
                TLOGNAME="TLOG"
                TLOGSIZE=100
                MAXACCESSERS=100
                MAXWSCLIENTS=50
#
*GROUPS
"GRP1"      LMID="SITE1"   GRPNO=4
#
*SERVERS
     "WSL"  SRVGRP="GRP1"  SRVID=2
            CLOPT=" – -n //wsl.beasys.com:3107 -d /dev/tcp"
            RQPERM=0660    REPLYQ=Y RPPERM=0660
            MIN=1          MAX=1    CONV=N
            SYSTEM_ACCESS=FASTPATH
            MAXGEN=5       GRACE=86400   RESTART=Y
#
```

CHAPTER

3 BEA TUXEDO Workstation for UNIX System Workstations

What This Chapter Is About

This chapter covers the use of the Workstation on UNIX workstations. The material is organized under the following section headings:

- Coding and building clients
- System-delivered clients
  - `wmio`(1)
  - `wud`(1)
- Running BEA TUXEDO system clients on a UNIX workstation
  - Directory structure to support Workstation clients
  - Environment variables for Workstation clients

Coding and Building Clients

The source code for client programs running on UNIX workstations is the same as that for client programs within the BEA TUXEDO system administrative domain (that is, native clients). You have available all of the ATMI functions, FML functions, and the BEA TUXEDO system data entry system for defining and managing forms.

References to Other Guides

The *BEA TUXEDO Programmer's Guide* covers the use of the ATMI calls in considerable detail. There is a separate chapter dealing with client programs.

The *BEA TUXEDO FML Programmer's Guide* and the *BEA TUXEDO Data Entry System Guide* are useful for information on those two special subjects. Sections 1 and 3c of the *BEA TUXEDO Reference Manual* are the final authority for all BEA TUXEDO system commands and functions.

Building Clients

Workstation client programs are compiled and linked with the `buildclient(1)` command. If you are building a Workstation client on a native node (that is, one on which the complete BEA TUXEDO system is installed), use the `-w` option. This specifies that the client should be built using the workstation libraries. On a native node, where both native and workstation libraries are present, the default is to use the native libraries. The `-w` option ensures that the correct libraries for a workstation client are used. On a workstation, where only the workstation libraries are present, it is not necessary to use the `-w`.

Listing 3-1 shows an example of the `buildclient`(1) command line on a native node.

Listing 3-1 buildclient Command Line

```
TUXDIR=/var/opt/tuxedo CC=ncc; export TUXDIR CC
buildclient -w -o wsclt -f wsclt.c -f "userlib1.a userlib2.a"
```

The -o option provides a name for your executable file. Input files are specified with a -f *firstfiles* option in Listing 3-1 to indicate that they are called in ahead of system libraries. buildclient needs TUXDIR to locate system libraries. CC defaults to cc, but can be set to another compiler as in the example.

System-delivered Clients

Two system-delivered clients are available on UNIX workstations. These are standard BEA TUXEDO system clients that have been slightly modified for workstation use, which is shown by the w prefix.

wmio(1)
: manages mask input and output for Workstation clients on a UNIX workstation

wud(1)
: the BEA TUXEDO system driver program that sends FML buffers to BEA TUXEDO system servers

Application Password when Running from a Script

If the BEA TUXEDO system second-level security has been specified for an application, the system clients prompt the user for the application password. If the client program is being run from a script, which is a common occurrence with wud, the password is read from the APP_PW environment variable. Do not confuse the environment variable with the similar configuration file parameter, SECURITY, for which the value APP_PW enables the security feature.

Running BEA TUXEDO System Clients on a UNIX Workstation

After the client programs have been developed and tested, they can be moved to the workstations where they will be available to users.

Directory Structure to Support Workstation Clients

`APPDIR`
: client executables are commonly kept in the directory from which the application is run

`$TUXDIR/include`
: BEA TUXEDO system header files such as `atmi.h`

`$TUXDIR/bin`
: BEA TUXEDO system commands and system clients such as `wmio` and `wud`

`$TUXDIR/locale/C`
: default message catalog directory

`$TUXDIR/lib`
: runtime libraries if your application uses shared objects

Environment Variables

Workstation clients make use of several environment variables. The following are checked by `tpinit()` when the workstation client attempts to join the application:

`WSENVFILE`
: names a file containing environment variable settings to be set in the client's environment. The format of this file is described in further detail below.

`WSNADDR`
: Specifies the network address of the workstation listener process through which clients gain access to the application.

TCP/IP addresses may be specified in the following forms:

"*//host.name:port_number*"
"*//#.#.#.#:port_number*"

Note: If you are specifying TCP/IP addresses for a Windows NT workstation, omit the quotation marks.

In the first format, the domain finds an address for *hostname* using the local name resolution facilities (usually DNS). *hostname* must be the local machine, and the local name resolution facilities must unambiguously resolve *hostname* to the address of the local machine.

In the second example, the "*#.#.#.#*" is in dotted decimal format. In dotted decimal format, each #should be a number from 0 to 255. This dotted decimal number represents the IP address of the local machine.

In both of the above formats, *port_number* is the TCP port number at which the domain process will listen for incoming requests. *port_number* can be either a number between 0 and 65535 or a name. If *port_number* is a name, then it must be found in the network services database on your local machine.

The address can also be specified in hexadecimal format when preceded by the characters "0x". Each character after the initial "0x" is a number between 0 and 9 or a letter between A and F (case insensitive). The hexadecimal format is useful for arbitrary binary network addresses such as IPX/SPX or TCP/IP.

The address can also be specified as an arbitrary string. The value should be the same as that specified for the -n *netaddr* option of the wsl(5).

You can specify more than one address by including a comma-separated list of pathnames for WSNADDR. Addresses are tried in order until a connection is established. You can specify any item in an address list as a parenthesized grouping of pipe-separated network addresses. For example,

```
WSNADDR="(//m1.acme.com:3050|//m2.acme.com:3050), //m3.acme.com:3050"
```

The BEA TUXEDO system randomly selects one of the parenthesized addresses. This strategy distributes the load randomly across a set of listener processes. If this is being specified for a Windows NT workstation, leave off the quotation marks. Addresses are tried in order until a connection is established.

WSDEVICE
: names the device that accesses the network and is only required when the BEA TUXEDO system is using the TLI networking interface. In Release 6.4 (or higher), this variable is never required. In prior releases, it is required while using the SPX protocol under DOS or Windows. It should be set to `/dev/tcp` for TCP/IP and to `/dev/nspx` for the SPX networking protocol.

WSTYPE
: is used within `tpinit`() when invoked by a workstation client to negotiate encode/decode responsibilities with the native site. An unspecified `WSTYPE` always causes encoding, even if it is also unspecified on the native site. The only way to ensure that encode/decode is turned off is to explicitly specify the same `WSTYPE` value for both sites.

WSRPLYMAX
: is used by `tpinit`() to set the maximum amount of core memory that ATMI functions use for buffering application replies before they are dumped to disk. Replies that are not being waited for (using `tpgetrply`()), and unsolicited messages are buffered in this area. When this area gets filled with one or more messages, the overflow is written to a disk file. The system default limit for this is 32,000 bytes. The available memory on your machine is the key factor in deciding whether you should use `WSRPLYMAX` to set a lower limit. Writing replies to disk causes a substantial reduction in performance.

Other environment variables may be needed by Workstation clients on a UNIX workstation depending on what BEA TUXEDO system features are being used. The `compilation`(5) reference page recaps which variables are needed under what circumstances.

Environment File

When tpinit() is called, an environment file is read if it exists. Listing 3-2 shows a sample file that could be used for two different applications.

Listing 3-2 Environment File

```
TUXDIR=/opt/tuxedo
[application1]
;this is a comment
/* this is a comment */
#this is a comment
//this is a comment
set FIELDTBLS=app1_flds
set FLDTBLDIR=/opt/app1/udataobj
[application2]
FIELDTBLS=app2_flds
FLDTBLDIR=/opt/app2/udataobj
```

The format of the file is as follows:

- Any leading space and tab characters on each line are ignored and are not considered in the following points.
- Lines containing variables to be put into the environment are of the following form:

  variable=value
  or
  set variable=value

  where *variable* must begin with an alphabetic or underscore character and contain only alphanumeric or underscore characters, and *value* may contain any character except newline.
- Within the *value*, strings of the form ${*env*} are expanded using variables already in the environment. Forward referencing is not supported and if a value is not set, the variable is replaced with the empty string. Backslash (\) may be used to escape the dollar sign and itself. All other shell quoting and escape mechanisms are ignored and the expanded *value* is placed into the environment.

- Lines beginning with slash (/), pound sign (#), or exclamation point (!) are treated as comments and ignored. Lines beginning with other characters besides these comment characters, a left square bracket, or an alphabetic or underscore character are reserved for future use; their use is undefined.
- The file is partitioned into sections by lines of the form

  `[`*`label`*`]`

  where *`label`* is the name of the section and follows the same rules for *variable* above. The label is silently truncated if longer than 31 characters.
- Variable lines between the top of the file and the first label are put into the environment for all applications; this is the global section. A label of `[]` also indicates the global section. Other variables are put into the environment only if the label matches the application label specified for the application.

Using tuxreadenv

The function `tuxreadenv`(3c) can be used to read the environment file for a particular label:

```
void tuxreadenv(char *file, char *label)
```

If *`file`* is NULL, then a default file name is used. The default file names for various platforms are as follows:

DOS, Windows, OS/2, Windows NT
: `C:\TUXEDO\TUXEDO.ENV`

Macintosh
: `TUXEDO.ENV` in the system directory

NETWARE
: `SYS:SYSTEM\TUXEDO.ENV`

POSIX
: `/usr/tuxedo/TUXEDO.ENV` or `/var/opt/tuxedo/TUXEDO.ENV`

If the value of *`label`* is NULL, then only variables in the global section are put into the environment. For other values of *`label`*, the global section variables plus any variables in a section matching the *`label`* are put into the environment.

An error message is printed to the userlog if there is a memory failure, if a non-null file name does not exist, or if a non-null label does not exist.

Each time `tpinit`() is called (either explicitly or implicitly by calling another ATMI function), `tuxreadenv`() is called automatically in Workstation clients. If `WSENVFILE` is set in the environment, then it designates the environment file; otherwise, NULL is passed to `tuxreadenv`() for the file name so that the default file is used. If `WSAPP` is set in the environment, then it is to be used as the section label in the environment file; otherwise, NULL is passed to `tuxreadenv`() for the label name. Application clients may also call `tuxreadenv`() explicitly.

The environment is implemented and available in different ways on different platforms. A uniform interface to the environment is provided via the existing `tuxgetenv(3c)` and `tuxputenv(3c)` functions. These functions provide access to (a) all variables from the specified `WSENVFILE` file for the specified `WSAPP` label (or the defaults if not specified), and (b) the environment variables in the operating system environment.

4 BEA TUXEDO Workstation for MS-DOS Workstations

What This Chapter Is About

This chapter covers the use of the BEA TUXEDO Workstation feature on MS-DOS workstations. The material is organized under the following section headings:

- Coding and building clients
- System-delivered clients
  - `wud`(1)
- Running BEA TUXEDO clients on an MS-DOS workstation
  - Directory structure to support Workstation clients
  - Environment variables for Workstation clients

Prerequisites

The BEA TUXEDO System/Workstation for MS-DOS requires a workstation running MS-DOS 3.21 or later and a minimum of 1M of RAM.

The software to support network communications must be installed. The Workstation has been tested with the Sockets interface provided in the Novell LAN Workplace for DOS version 4.2 and compiled using the LAN Workplace Toolkit for DOS version 4.1 over TCP/IP. The toolkit is only required while developing applications and not needed for a runtime system.

The SPX software has been tested using the NetWare client and compiled using the NetWare client SDK version 1.0. The NetWare client SDK is only required while developing applications and not needed for a runtime system.

The libraries are compiled using the Microsoft C++ Version 1.5, using the Large memory model. The Medium memory model is no longer supported.

Coding and Building Clients

Client programs for MS-DOS workstations are the same as client programs within the BEA TUXEDO system administrative domain, with one exception:

- The BEA TUXEDO system data entry system and the form handler `mio`(1) are not available

You do have available all of the ATMI functions. You have all FML functions with the following exceptions (and their VIEW counterparts):

`Fboolco()` `Fboolev()` `Fboolpr()` `Ffloatev()`

The FML function `Ffindocc` is present but is not able to search for regular expressions. The external variable `Ferror` is redefined to `FMLerror`.

Buffer Size Limitation

On DOS and WINDOWS workstations, message buffer sizes are limited to 64K less the size of the message header (currently under 400 bytes).

References to Other Guides

The *BEA TUXEDO Programmer's Guide* covers the use of the ATMI calls in considerable detail. There is a separate chapter dealing with client programs.

The *BEA TUXEDO FML Programmer's Guide* is useful for information on the Field Manipulation Language. Reference pages in the *BEA TUXEDO Reference Manual* are the final authority for all BEA TUXEDO system commands and functions.

Building Clients

Workstation client object files are link edited with the `buildclt`(1) command. While the syntax of the command is straightforward, the usage varies according to the compilation system used.

buildclt syntax

`buildclt` has the following options:

`-o` *name*
: specifies the file name of the executable file being created. The default is `client.exe`.

`-c {m | i}`
: specifies the compilation system to be used. `m` stands for Microsoft compilation system. `i` stands for the IBM OS/2 Cset2 compiler and is only used on OS/2 32 bit. Microsoft is the default. The system specified imposes other requirements discussed below.

`-m {m | l}`
: specifies the `m`edium or `l`arge memory model. The default is `l`. The medium memory model is not supported under DOS.

`-f` *`firstfiles`*
: indicates one or more object files to be included before the BEA TUXEDO system libraries. `-f` can also be used to pass options to the compiler or linker. If more than one file name is specified, the names are separated by white space and the list is enclosed in quotation marks. The `-f` option can appear more than once.

`-l` *`libfiles`*
: specifies libraries to be included after the BEA TUXEDO system libraries. If more than one file name is specified, the names are separated by white space and the list is enclosed in quotation marks. The `-l` option can appear more than once.

`-d` *`deffile`*
: specifies a module definition file used for linking a Windows program.

`-W`
: indicates a Windows or Windows NT client is being built.

`-O`
: indicates an OS/2 client is being built.

Microsoft Compilation Environment

The Microsoft C environment expects to find library directory names in the environment variables `INCLUDE` and `LIB`. They might be set as follows:

```
INCLUDE=C:\C600\INCLUDE;C:\TUXEDO\INCLUDE
LIB=C:\NET\TOOLKIT\LIB;C:\C600\LIB;C:\TUXEDO\LIB
```

The `C:\NET` directory is the location of the LAN Workplace Toolkit.

`buildclt` can be used to link edit the executable. The client source files should be compiled separately using the `CL` command. The `-f` option is used to pass options to the `LINK` command, as well as to name the input object files.

Examples of possible options are:

```
-f /SE:200     # to set the number of segments used to 200
-f /ST:10000   # to set default stack size to 10000 bytes
-f /CO         # to create a file that can be debugged by Codeview
              # (assumes the file was compiled with the -Zi option)
```

buildclt Examples

Listing 4-1 shows some sample `buildclt` command lines. All of the examples show an executable named `emp.exe` being built.

Listing 4-1 Example of buildclt(1) command lines

```
DOS:
buildclt -cm -ml -o emp.exe -f "/CO/ST:10000/SE:200" -f emp.obj -l llibsock.lib
WINDOWS:
buildclt -W -cb -mm -o emp.exe -f "-v emp.obj" -l twlbsock.lib -d emp.def
buildclt -W -cm -mm -o emp.exe -f "/CO emp.obj" -l wlibsock.lib -d emp.def
```

System-delivered Client

A system-delivered client is available on MS-DOS workstations. This is a standard BEA TUXEDO system client that has been slightly modified for workstation use, which is shown by the `w` prefix.

`wud`(1)
: The BEA TUXEDO driver program that sends `FML` buffers to BEA TUXEDO system servers

Application Password when Running from a Script

If BEA TUXEDO second-level security has been specified for an application, the system clients prompt the user for the application password. If the client program is being run from a script, which is a common occurrence with `wud`, the password is read from the `APP_PW` environment variable.

Running BEA TUXEDO System Clients on a Workstation

After the client programs have been developed and tested, they can be moved to the MS-DOS workstations where they will be available to users.

Directory Structure to Support /WS Clients

The following directories should be present under the directory designated as `TUXDIR`.

`bin`
: BEA TUXEDO system commands and system clients, for example, `wud`

`locale/C`
: international message directory (English version). Other language directories may be present under `locale`.

Environment Variables

Workstation clients make use of several environment variables. The following are checked by `tpinit()` when the client attempts to join the application:

`WSENVFILE`
: Names a file containing environment variable settings to be set in the client's environment. All of the other environment variables needed by client programs can be contained in this file. The format of this file is described in further detail below.

`WSNADDR`
: Specifies the network address of the workstation listener process through which clients gain access to the application.

  TCP/IP addresses may be specified in the following forms:

  "//host.name:port_number"
  "//#.#.#.#:port_number"

Note: If you are specifying TCP/IP addresses for a Windows NT workstation, omit the quotation marks.

In the first format, the domain finds an address for *hostname* using the local name resolution facilities (usually DNS). *hostname* must be the local machine, and the local name resolution facilities must unambiguously resolve *hostname* to the address of the local machine.

In the second example, the "#.#.#.#" is in dotted decimal format. In dotted decimal format, each # should be a number from 0 to 255. This dotted decimal number represents the IP address of the local machine.

In both of the above formats, *port_number* is the TCP port number at which the domain process will listen for incoming requests. *port_number* can either be a number between 0 and 65535 or a name. If *port_number* is a name, then it must be found in the network services database on your local machine.

The address can also be specified in hexadecimal format when preceded by the characters "0x". Each character after the initial "0x" is a number between 0 and 9 or a letter between A and F (case insensitive). The hexadecimal format is useful for arbitrary binary network addresses such as IPX/SPX or TCP/IP.

The address can also be specified as an arbitrary string. The value should be the same as that specified for the NLSADDR parameter in the NETWORK section of the configuration file.

More than one address can be specified if desired by specifying a comma-separated list of pathnames for WSNADDR. Addresses are tried in order until a connection is established. Any member of an address list can be specified as a parenthesized grouping of pipe-separated network addresses. For example,

```
WSNADDR="(//m1.acme.com:3050|//m2.acme.com:3050), //m3.acme.com:3050"
```

The BEA TUXEDO system randomly selects one of the parenthesized addresses. This strategy distributes the load randomly across a set of listener processes. Addresses are tried in order until a connection is established.

WSTYPE
: is used within tpinit() when invoked by a workstation client to negotiate encode/decode responsibilities with the native site. An unspecified WSTYPE always causes encoding, even if it is also unspecified on the native site. The only way to ensure that encode/decode is turned off is to specify the same WSTYPE value for both sites.

WSDEVICE
: In Release 6.4 or higher, this variable is never required. In prior releases, it is required while using the SPX protocol under DOS or Windows. It should be set to /dev/nspx for the SPX networking protocol.

WSRPLYMAX
: is used by tpinit() to set the maximum amount of core memory that ATMI functions use for buffering application replies before they are dumped to disk. Replies that are not being waited for (using tpgetrply()), and unsolicited messages are buffered in this area. When this area gets filled with one or more messages, the overflow is written to a disk file. The system default limit for this is 32,000 bytes. The available memory on your machine is the key factor in deciding whether you should use WSRPLYMAX to set a lower limit. Writing replies to disk causes a substantial reduction in performance.

Other environment variables may be needed by Workstation clients on an MS-DOS workstation depending on what BEA TUXEDO system features are being used. Two MS-DOS conditions apply:

- Environment variables that take directory lists, for example, FLDTBLDIR and VIEWDIR, must observe MS-DOS conventions.
  - Directory names are separated by semicolons rather than colons.
  - Pathname components are separated by backslashes rather than slashes.
  - The default field table file looked for by mkfldhdr(1) is fld.tbl in the UNIX environment. The default output file is fld.tbl.h, which is an illegal file name under MS-DOS. In the MS-DOS environment, the default names are fldtbl and fldtbl.h.
- Binary view description file names are given a .vv suffix rather than the single .V of the case-sensitive UNIX environment.

The compilation(5) manual page recaps which variables are needed under what circumstances.

When `tpinit()` is called, an environment file will be read if it exists. Here is an example file that could be used for two different applications.

```
TUXDIR=C:\TUXEDO
[application1]
;this is a comment
/* this is a comment */
#this is a comment
//this is a comment
SET FIELDTBLS=APP1_FLDS
set FLDTBLDIR=C:\APP1\UDATAOBJ
[application2]
FIELDTBLS=APP2_FLDS
FLDTBLDIR=C:\APP2\UDATAOBJ
```

The format of the file contents is as follows:

- Any leading space and tab characters on each line are ignored and are not considered in the following points.
- Lines containing variables to be put into the environment are of the form

  variable=*value*
  or
  set *variable*=*value*

  where *variable* must begin with an alphabetic or underscore character and contain only alphanumeric or underscore characters, and *value* may contain any character except newline.
- Within the *value*, strings of the form ${*env*} are expanded using variables already in the environment (forward referencing is not supported and if a value is not set, the variable is replaced with the empty string). Backslash (\) may be used to escape the dollar sign and itself. All other shell quoting and escape mechanisms are ignored and the expanded *value* is placed into the environment.
- Lines beginning with slash (/), pound sign (#), or exclamation point (!) are treated as comments and ignored. Lines beginning with other characters besides these comment characters, a left square bracket, or an alphabetic or underscore character are reserved for future use; their use is undefined.
- The file is partitioned into sections by lines of the form

  [label]

  where *label* is the name of the section and follows the same rules for *variable* above. The label will be silently truncated if longer than 31 characters.

- Variable lines between the top of the file and the first label are put into the environment for all applications; this is the global section. A label of `[]` will also indicate the global section. Other variables are put into the environment only if the label matches the application label specified for the application.

The following function can be used to read the environment file for a particular label.

```
void tuxreadenv(char *file, char *label)
```

If *file* is NULL, then a default file name is used. The default file names for various platforms are as follows:

DOS, Windows, OS/2
: `C:\TUXEDO\TUXEDO.ENV`

Macintosh
: `TUXEDO.ENV` in the system preferences directory

NETWARE
: `SYS:SYSTEM\TUXEDO.ENV`

POSIX
: `/usr/tuxedo/TUXEDO.ENV` or `/var/opt/tuxedo/TUXEDO.ENV`

If *label* is NULL, then only variables in the global section are put into the environment. For other values of *label*, the global section variables plus any variables in a section matching the *label* are put into the environment.

An error message is printed to the userlog if there is a memory failure, if a non-null file name does not exist, or if a non-null label does not exist.

Each time `tpinit()` is called (either explicitly or implicitly by calling another ATMI function), `tuxreadenv()` will be called automatically in Workstation clients. If `WSENVFILE` is set in the environment, then it will be used as the name of the environment file; otherwise, NULL will be passed to `tuxreadenv()` for the file name such that the default file is used. If `WSAPP` is set in the environment, then it will be used as the section label in environment file; otherwise, NULL will be passed to `tuxreadenv()` for the label name. Application clients may also call `tuxreadenv()` explicitly.

The environment is implemented and available differently on different platforms. A uniform interface to the environment is provided via the existing `tuxgetenv`(3c) and `tuxputenv`(3c) functions. These functions provide access to all variables set both from the specified `WSENVFILE` file for the specified `WSAPP` label (or the defaults if not specified) and the environment variables set via the operating system environment, if supported (in the case of MS Windows, the DOS environment is available).

5 BEA TUXEDO Workstation for WINDOWS

What This Chapter Is About

This chapter describes the use of the BEA TUXEDO Workstation for Microsoft Windows, Version 3.0 and later.

This instantiation offers significant benefits to the application developers:

- More memory is available to Windows applications, as compared to MS-DOS applications which normally cannot access memory above 640K.
- Executable text is shared among applications, saving memory.
- BEA TUXEDO Workstation upgrades are possible without relinking or modifying an application program's executable file.
- Dynamic linking permits interpretive graphical application generator tools (such as Visual Basic, ObjectVision and SQLWindows) to call BEA TUXEDO system services.

The major sections in this chapter cover:

- Software prerequisites
- BEA TUXEDO client programs and the Windows DLL.

Definitions of Terms, Acronyms, and Abbreviations

BEA TUXEDO system terms are defined in the *BEA TUXEDO Glossary*, but we have extracted terms specific to this feature:

Dynamic Link Libraries (DLL)
: A DLL is a collection of functions grouped into a load module that is dynamically linked with an executable program at run time. It is similar to a shared object under the UNIX operating system. The Microsoft Windows operating environment makes extensive use of this feature. Most software products for Windows provide a DLL interface.

Ordinal Export Numbers
: An ordinal export number is a number assigned in a module definition file, by which a function in a DLL may be referenced. Some software packages reference DLL functions by name, while others (for example, SQLWindows) reference DLL function entry points by numbers only.

Import libraries
: An import library is a collection of stub function names associated with a DLL. To link edit an application object calling a DLL routine, the linker needs a stub that defines where the subroutine exists.

Module Definition File
: This file defines the characteristics of an executable and is used at link time. For a DLL this file details the exported functions along with their import numbers that can be called from the DLL and other imported functions which the DLL will call.

Prerequisites

This section lists the hardware and software prerequisites.

Hardware

The BEA TUXEDO Workstation for Windows runs on Intel 80286 processors and above.

The machine on which the BEA TUXEDO Workstation is installed runs as a remote machine to a UNIX server.

Software

Workstation for Windows runs under the Windows Version 3.0 or later operating system.

In Windows while using TCP/IP any Windows Sockets Compliant TCP/IP stack can be used. The Windows DLL has been tested with the NOVELL LAN WorkPlace for MS-DOS using the Windows Sockets interface.

In Windows 95 the native TCP/IP stack is used.

The UNIX server machine must have the BEA TUXEDO system and the native-side BEA TUXEDO Workstation installed.

Programming Considerations with the Windows DLL

This section covers items specific to writing and building BEA TUXEDO client programs to run under Microsoft Windows. They are intended to supplement the material presented in the *BEA TUXEDO Programmer's Guide.*

For information on defining application-specific buffers types for the Windows environment, please see *Administering the BEA M3 System* and `tuxtypes`(5) in the *BEA TUXEDO Reference Manual.*

Our assumption is that readers of this section either have experience in writing Windows programs or have access to tutorial material on that subject. Our discussion is limited to a description of how you go about putting BEA TUXEDO Workstation functions into Windows modules.

Writing Client Programs

The ATMI and FML calls used in Windows client programs are much the same as described in the *BEA TUXEDO Programmer's Guide.* They must, however, be incorporated into Windows modules. The following things work slightly differently than they do in the UNIX environment.

Global Variables

The error global variables are not available in the way they normally are; they are defined as macros in the `.h` files. To make them available in client programs:

```
for tperrno or tpurcode -- #include "atmi.h"
for Ferror -- #include "fml.h"
for Uunixerr -- #include "Uunix.h"
for proc_name -- #include "userlog.h"
```

See Listing 5-2 for an example.

`tpsetunsol`

Use `MakeProcInstance()` to create a "thunk" pointer. Both the `.DEF` file and the `.C` file must include the definition. This step is required under Windows 3.x.

See Listing 5-4 for an example.

Buffer Size

On DOS and WINDOWS workstations, message buffer sizes are limited to 64K less the size of the message header (currently under 400 bytes).

Environment

Once set, the environment in MS Windows cannot be changed. Because of this the environment will be handled locally within the DLL. The functions `tuxgetenv`(3) and `tuxputenv`(3) have been created for this purpose. An example is given in Listing 5-1; the reference pages are in the *BEA TUXEDO Reference Manual*.

Listing 5-1 Handling the Environment

```
LPSTR szRplyMax;
long wCurrRplyMax = 0; /* Current value of BEA TUXEDO Reply buffer
                        * in memory
                        */
....
if(szRplyMax = tuxgetenv((LPSTR)"WSRPLYMAX") == NULL)
               wCurrRplyMax = 0L;
else
               wCurrRplyMax = atol(szRplyMax);
if(wCurrRplyMax  10240L) {
               if(tuxputenv("WSRPLYMAX=16384") == 0)
                       wCurrRplyMax = 16384L;
}
```

Using bankapp as an Example

Among the files in the directory $TUXDIR/apps/ws you will find the following.

```
BALANCE.DLG  - dialog resource for BANKAPPW BALANCE window
BANKAPPW.C   - Windows application
BANKAPPW.DEF - Windows definition file for BANKAPPW
BANKAPPW.RC  - Windows resource file for BANKAPPW
BANKAPP.H    - BANKAPPW window field identifiers
BANKFLDS     - FML buffer field identifiers
NT.MAK       - Microsoft C makefile
```

These are the files needed to produce a bankapp client for Windows.

Take a look through the application file, BANKAPPW.C; we want to call your attention to a few items. Listing 5-2 shows the #include files you should include.

Listing 5-2 bankapp for Windows: #include Files

```
15   #include stdio.h>
16   #undef NULL
17   #include windows.h>
18   #include stdlib.h>
19   #include string.h>
20   #include ctype.h>
21   #include atmi.h>
22   #include Usysflds.h>
23   #include fml.h>
24   #include userlog.h>
25   #include "bankapp.h"
26   #include "bankflds.h"
```

The #undef NULL at line 16 is there to prevent a compiler warning caused by NULL being defined in both stdio.h and windows.h.

At line 45, as Listing 5-3 shows, you declare your FBFR as a FAR pointer. In Windows NT FAR and PASCAL are defined to be nothing. Listing 5-3 also shows that you write your client program as a WinMain() module.

Listing 5-3 bankapp for Windows: WinMain Declaration

```
44   static HANDLE   hInst;
45   static FBFR FAR *fbfr;
46
47   int PASCAL
48   WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
49   HANDLE hInstance;
50   HANDLE hPrevInstance;
51   LPSTR lpCmdLine;
52   int nCmdShow;
```

In Listing 5-4 there are examples of the statements needed to declare a routine to pick up unsolicited messages. The statement must be in both the `.C` file and the `.DEF` file. The example also shows a call to `tpsetunsol`.

Listing 5-4 bankapp for Windows: Declaring Unsolicited Message Routine

```
In BANKAPPW.C

   40  BOOL FAR PASCAL CloseDlg(HWND, WORD, WORD, LONG);
   41  BOOL FAR PASCAL OpenDlg(HWND, WORD, WORD, LONG);
   42  void FAR PASCAL UnsolProc(char FAR *, long, long);
.
.
.
  167                    lpfnCloseDlg = MakeProcInstance(CloseDlg,hInst);
  168                    lpfnOpenDlg = MakeProcInstance(OpenDlg,hInst);
  169                   lpfnUnsolptr=MakeProcInstance((FARPROC)UnsolProc,hInst);
  170
  171                    if(tpsetunsol((void FAR*)lpfnUnsolptr) == TPUNSOLERR)
  172                             userlog("tpsetunsol failed");

In BANKAPPW.DEF
   16  EXPORTS           WndProc
   17                    TransferDlg
   18                    BalanceDlg
   19                    DepositDlg
   20                    WithdrawDlg
   21                    CloseDlg
   22                    OpenDlg
   23                    UnsolProc
```

Lines 183 to 325 are Field Validation Routines: five routines that check the validity of the input typed in by the user. We will not show this code except for an example (in Listing 5-5) of how syntax errors cause a message to be displayed on the user's screen by means of a call to `MessageBox`.

Listing 5-5 bankapp for Windows: Displaying a Syntax Error

```
203       if (i  5 || i > 6 || account[i] != '\0') {
204               /*SetDlgItemText (hDlg, item, ""); */
205                 MessageBox (hDlg,
206                 "Account number must be 5 or 6 digits",
207                 "BANKAPP", MB_OK);
208                 SetFocus(GetDlgItem(hDlg, item));
209                 return(-1);
```

The actual work of the application begins at with the comments at line 354 describing six dialog boxes. Listing 5-6 shows the beginning of this section of the code.

Listing 5-6 bankapp for Windows: Dialog Boxes

```
354  /*
355   *  Routines to handle dialog boxes for Services
356   *      BalanceDlg(hDlg, message, wParam, lParam)
357   *      CloseDlg(hDlg, message, wParam, lParam)
358   *      DepositDlg(hDlg, message, wParam, lParam)
359   *      OpenDlg(hDlg, message, wParam, lParam)
360   *      TransferDlg(hDlg, message, wParam, lParam)
361   *      WithdrawDlg(hDlg, message, wParam, lParam)
362   */
363  BOOL FAR PASCAL
364  BalanceDlg(hDlg, message, wParam, lParam)
365  HWND hDlg;
366  WORD message;
367  WORD wParam;
368  LONG lParam;
```

The dialog boxes, which take up the rest of the code, accept input from the user, start a transaction (assuming `-Dtran` is specified when the client is built), make a call to the requested service and return information to the user. The code in Listing 5-7 shows how errors might be handled. In lines 391-392, for example, the call to `tpbegin` fails, a message is sent to `userlog`(3c) and also to the user's screen via the string `account1`. In lines 397-403, if the service call fails and the buffer is not NULL, the status line is picked up and returned to the user. If the failure is due to another reason or if the buffer is NULL, a hard-coded error message is returned.

Lines 416-417 show the service request being successfully performed and the requested balance being displayed on the user's screen.

Listing 5-7 bankapp for Windows: Error Handling

```
390  #ifdef tran
391                  if (tpbegin(30, 0) == -1) {
392                          (void) userlog("failed to begin transaction\n");
393                          lstrcpy(account1, "Transaction failed");
394                  }
395                  else
396  #endif
397                  if (tpcall("INQUIRY", (char FAR *)fbfr, 0,
                                 (char FAR *FAR *)&fbfr, &len, 0) == -1) {
398                          if(tperrno== TPESVCFAIL && fbfr != NULL &&
399                          (s=Ffind(fbfr,STATLIN,0,0)) != 0) {
400                           lstrcpy(account1, s);
401                          }
402                          else
403                           lstrcpy(account1,"Inquiry failed");
404  #ifdef tran
405                          (void) tpabort(0);
406  #endif
407                  }
408                  else {
409  #ifdef tran
410                          if(tpcommit(0) < 0) {
411                                  lstrcpy(account1, "Inquiry failed");
412                          }
413                          else
414  #endif
415
416                          wsprintf(account1, "Account Balance:  %s",
417                                  (LPSTR) Ffind(fbfr, SBALANCE, 0, 0));
418                  }
```

Blocking Network Behavior

When an ATMI function is called the Windows DLL could block on the network waiting for a reply from the server on the UNIX machine. This can happen on any call that initiates a network message to the UNIX machine, for example, `tpcall()`, `tpinit()`, `tpgetrply()` and so on. These functions may take an arbitrary long time to complete; a good example is `tpcall()`, which may block until the server has completed the processing required.

Windows 3.x is not a preemptive multitasking operating system and there is only one system wide Windows input queue. Because of this if a client program blocks on the network, the entire Windows interface "freezes" until the network call returns.

With the BEA TUXEDO Workstation for Windows a blocking operation that cannot be completed immediately is handled as follows. The DLL initiates the operation and enters a loop in which it dispatches any windows messages (yielding the processor to another thread if necessary) and then checks for the completion of the ATMI function. If the ATMI call is complete the blocking call is completed and the appropriate result is returned to the caller. If not complete the DLL continues to dispatch Windows messages. For a complete description of this behavior see `AEWsetblockinghook`(3c) in the *BEA TUXEDO Reference Manual.*

If a Windows message is received for a process for which a blocking operation is in progress, there is a risk that the application will attempt to issue another ATMI call. Such application behavior is not supported by ATMI calls. `AEWisblocked`(3c) can be called at any time to detect if there is a blocking ATMI call outstanding. Any other ATMI call made while this condition exists will fail and set `tperrno` to `TPEPROTO`.

Although this mechanism is sufficient for simple applications, it cannot support the complex message dispatch requirements of more advanced applications (for example, those using the MDI (Multiple Document Interface) model). For such applications, ATMI includes `AEWsetblockinghook`(3c), which allows the programmer to define a special routine that will be called instead of the default routine.

If an application invokes a blocking operation like `tpcall()` and provides a typed buffer to it as an argument, it is the responsibility of the application to ensure that the buffer is available to ATMI until the operation is completed.

FML functions will continue to work even if there is a blocking call in progress, therefore it is the responsibility of the application to not use FML buffers that are passed in as an argument to an ATMI call that is currently in progress until the ATMI call completes.

Restoring the Environment

As Workstation for Windows 3.x starts up it copies the environment from the Windows area to a local buffer and maintains a distinct environment space for each client. The local space is destroyed when `tpterm()` is called. It is the responsibility of the application to reinstate the environment and other information installed using `tpsetunsol()` and `AEWsetblockinghook()` after `tpterm()` is called.

Building Client Programs

For Windows any compiler that can read Microsoft C import libraries can be used.

When compiling BEA TUXEDO client programs for Windows 3.x, use the C preprocessor flag

```
-D_TM_WIN
```

When link editing your client programs, use `buildclt`(1) with the `-W` flag.

BEA TUXEDO clients can be built without using the `buildclt`(1) utility. If the Microsoft Visual C++ projects are used then set the Preprocessor options as follows.

- For Windows 3.x
  `-D_TM_WIN`
- For Windows 95
  `-DWIN32`
- In the linker options:
  - For Windows 3.x, add `WTUXWS.LIB` to the input libraries.
  - For Windows 95, add `WTUXWS32.LIB MSVCRT.LIB` to the input libraries.

In addition to this set the `INCLUDE`, `LIB`, and `PATH` search directories appropriately.

Using views in 16-bit Windows

Windows Version 3.x (16 bit) recommends that C programs use "structure packing" on a one byte boundary. With the Microsoft C compiler this is enabled using the `/Zp` option. The view compiler `viewc` uses the Microsoft C compiler to calculate the offsets of the C structures. In order for `viewc` to use the structure packing and get the correct offsets, set the `CFLAGS` variable to `/Zp` before running `viewc`. As an example if the client C program uses structure packing on a one byte boundary then compile the C code as follows.

```
CL /Zp -D_TM_WIN CLIENT.C
and before compiling the views
set CFLAGS=/Zp
VIEWC.EXE CLVIEW.V
```

Runtime

When you run client programs, your `PATH` must include `%TUXDIR%/bin`.

Limitations

The following is a list of limitations that apply to Release 5.0 (and higher) of the BEA TUXEDO system Windows DLL:

- Microsoft Windows "real" mode is not supported under Windows 3.X.
- Multiplexed network connections are not available; each client process requires a separate network connection.
- The BEA TUXEDO libraries (DLLs) are not thread-safe. This means that either applications should not use threads, or threaded access is serialized through all BEA TUXEDO calls (such as ATMI, FML, `userlog()`, and so on).

6 BEA TUXEDO Workstation for OS/2

What This Chapter Is About

This chapter describes the BEA TUXEDO Workstation under the OS/2 operating system.

This instantiation offers significant benefits to application developers:

- More memory is available to OS/2 applications, as compared to MS-DOS applications which normally cannot access memory above 640K.
- Executable text is shared among applications, saving memory.
- Workstation upgrades are possible without relinking or modifying an application program's executable file.
- Dynamic linking permits interpretive graphical application generator tools (such as ObjectVision) to call BEA TUXEDO system services.

The major sections in this chapter cover:

- Software prerequisites
- BEA TUXEDO client programs and the OS/2 environments

Definitions of Terms, Acronyms, and Abbreviations

BEA TUXEDO system terms are defined in the *BEA TUXEDO Glossary*, but we have extracted terms specific to this feature.

Dynamic Link Libraries (DLL)
: A DLL is a collection of functions grouped into a load module that is dynamically linked with an executable program at run time. It is similar to a shared object under the UNIX operating system. The IBM OS/2 operating environment makes extensive use of this feature. Most software products for OS/2 provide a DLL interface.

Ordinal Export Numbers
: An ordinal export number is a number assigned in a module definition file, by which a function in a DLL may be referenced. Some software packages reference DLL functions by name, while others (for example, SQLWindows) reference DLL function entry points by numbers.

Import Libraries
: An import library is a collection of stub function names associated with a DLL. To link edit an application object calling a DLL routine, the linker needs a stub that defines where the subroutine exists.

OS/2 Character Mode
: OS/2 Character mode is the character-based non-graphical user interface under OS/2.

Module Definition File
: This file defines the characteristics of an executable and is used at link time. For a DLL this file details the exported functions that can be called from the DLL and other imported functions which the DLL will call.

Windows Emulation Mode

It is possible for OS/2 version 2.0 to execute Windows applications, which in turn could call the BEA TUXEDO system's Windows DLL. LAN WorkPlace for OS/2 permits BEA TUXEDO system's Windows DLL to execute in this environment.

Prerequisites

This section lists the hardware and software prerequisites.

Hardware

The BEA TUXEDO Workstation DLL for OS/2 runs on Intel 80386 and above processors.

The machine on which the DLL is installed runs as a remote machine to a BEA TUXEDO system machine.

Software

The OS/2 DLL runs under the IBM OS/2 2.0 operating system for OS/2 character mode applications.

The BEA TUXEDO Workstation provides a 32-bit DLL for OS/2 that supports the OS/2 2.x version of the operating system. The networking software for this version is IBM TCP/IP. For developing applications a 32-bit compiler can be used. The DLL was compiled using the IBM Cset 2 C++ compiler, which is also the reference compiler.

The BEA TUXEDO system server machine must have the BEA TUXEDO system Release 4.2 (or higher) and the BEA TUXEDO Workstation installed.

Programming Considerations with OS/2 Clients

This section covers items specific to writing and building BEA TUXEDO client programs to run under OS/2. This material is intended to supplement the material presented in the *BEA TUXEDO Programmer's Guide*.

Writing Client Programs

The ATMI and FML calls used in OS/2 client programs are much the same as described in the *BEA TUXEDO Programmer's Guide*. They must, however, be incorporated into OS/2 modules. The following things work slightly differently than they do in the UNIX environment.

Global Variables

The error global variables are not available in the way they normally are; they are defined as macros in the `.h` files. To make them available in client programs.

```
for tperrno or tpurcode – #include "atmi.h"
for Ferror – #include "fml.h"
for Uunixerr – #include "Uunix.h"
for proc_name – #include "userlog.h"
```

OS/2 Character Mode

There is very little difference between writing C code for BEA TUXEDO client programs in this environment and writing them in the UNIX environment. For information on the ATMI calls, please refer to the *BEA TUXEDO Programmer's Guide*.

The special aspect of both MS-DOS and OS/2 character mode client programs is that you probably have to provide an application-specific form and menu handler. You might find it useful to look through the first two-thirds of the sample program,

BANKAPP.C, provided with the BEA TUXEDO system. This program serves as the client program for both environments (the .MAK files copy BANKAPP.C to BANKAPPO.C for OS/2, and it is built with a -O flag and different libraries). This part of the sample program is an MS-DOS/OS/2 form and menu handler. If you don't already have similar (or more sophisticated) interfaces at your installation, you may want to adapt this for your application.

The final third of BANKAPP.C (beginning at line 805) is the bankapp application client.

Building Client Programs

For the 32 bit platform the IBM Cset 2 C++ compiler is supported.

When compiling BEA TUXEDO client programs, use the C preprocessor flag

```
-D_TM_OS2
```

When link editing your client programs, use buildclt(1) with the -O flag for OS/2 character mode clients.

The buildclt(1) utility also supports the Microsoft C and the IBM compiler. Use the -c option to specify the compiler type: "m" for microsoft and "i" for IBM.

If you want to use the C compiler instead of buildclt(1) then use the C preprocessor flag

```
-D_TM_OS2
```

and while linking specify OTUXWS.LIB as an input library for OS/2.

Runtime

When you run client programs, your PATH and LIBPATH must include $TUXDIR/bin.

Limitations

The following is a list of limitations that apply to the present release of the BEA TUXEDO system OS/2 DLL.

- Multiplexed network connections are not available; each client process requires a separate network connection.
- The BEA TUXEDO Workstation DLL does not support threads.

7 BEA TUXEDO Workstation for Macintosh

What This Chapter Is About

This chapter describes the installation and use of the BEA TUXEDO Workstation for Apple Macintosh, System 7.1.

This instantiation offers developers the ability to write application clients using the Macintosh user interface.

The major sections in this chapter cover:

- Software prerequisites
- BEA TUXEDO client programs and the Macintosh

Prerequisites

This section lists the hardware and software prerequisites.

Hardware

The BEA TUXEDO Workstation for Macintosh runs on Motorola 68020, 68030 and 68040 processors.

The machine on which the BEA TUXEDO Workstation is installed runs as a remote machine to a UNIX server.

Software

- Macintosh System 7.1 or later
- Macintosh TCP/IP Version 2.0.6 or later
- If you are using the Symantec Think C Development Environment (68K only): Symantec Think C Version 7.0 or later
- If you are using the MetroWerks Code Warrior Development Environment (PowerPC only):
  - MetroWerks Code Warrior Version 9.0 or later
  - Macintosh Programmer's Workshop tools (included on the MetroWerks CD)

The BEA TUXEDO server machine must have the BEA TUXEDO system and the native-side BEA TUXEDO Workstation installed.

Programming Considerations with the Macintosh Libraries

This section covers items specific to writing and building BEA TUXEDO Workstation client programs to run under Macintosh. This material is intended to supplement the material presented in the *BEA TUXEDO Programmer's Guide*.

Our assumption is that readers of this section either have experience in writing Macintosh programs or have access to tutorial material on that subject. Our discussion is limited to a description of how you go about putting BEA TUXEDO functions into Macintosh programs.

Writing Client Programs

The ATMI and FML calls used in Macintosh client programs are much the same as described in the *BEA TUXEDO Programmer's Guide*. An example of `tuxputenv`(3c) is given in Listing 7-1.

Listing 7-1 Handling the Environment

```
void
main()
{
            TPINIT      *tpinfop;
            FILE        *fp;
            char        aline[MAXLINE];
            char        *loc_env[MAXENV];
....
          if((loc_env[top_one]=(char *) malloc(strlen(aline) + 1)) == NULL) {
             userlog("Error mallocing space for the environment.");
             continue;
          }
          strcpy(loc_env[top_one], aline);
          if(tuxputenv(loc_env[top_one]) != 0) {
             userlog("Error adding %s to the environment.",
             loc_env[top_one]);
             free(loc_env[top_one]);
             continue;
          }
```

Using bankapp as an Example

After the BEA TUXEDO system is installed on the Macintosh, among the files in the directory `$TUXDIR/apps/bankapp` you will find the following:

```
bankapp.r     - dialog resource for BANKAPPM windows
BANKAPPM.C    -   Macintosh application source file
BANKAPPM.h    - Macintosh application header file
MACDialog.c   - Macintosh application source file
MACDialog.h   - Macintosh application header file
bank.flds     - FML buffer field identifiers
mkbankhdr.tsh - TuxShell script to create headers
```

These are the files needed to produce a `bankapp` client for Macintosh. Notice there is a `TuxShell`(1) script, `mkbankhdr.tsh`, which needs to be modified for your instantiation and run with the `TuxShell`(1) program before compilation.

Take a look through the application file, `BANKAPPM.c`; we want to call your attention to a few items. Listing 7-2 shows a partial list of the `#include` files you should use.

Listing 7-2 bankapp for Macintosh: #include Files

```
41     #include <Quickdraw.h>
42     #include <Resources.h>
43     #include <Fonts.h>
44     #include <OSEvents.h>
45     #include <Desk.h>
46     #include “MACDialog.h”
47     #include “BANKAPPM.h”
48     #include “bank.flds.h”
49     #include <string.h>
50     #include <Uunix.h>
51     #include <atmi.h>
52     #include <fml.h>
53     #include <Usysflds.h>
54     #include <userlog.h>
```

Lines 93 to 410 (not shown) are Field Validation Routines: five routines that check the validity of the input typed in by the user. We will not show this code except for an example (in Listing 7-3) of how syntax errors cause a message to be displayed on the user’s screen by means of a call to `NoteAlert`.

Listing 7-3 bankapp for Macintosh: Displaying a Syntax Error

```
151  for (i=0; amount[i] != 0 & i 10; i++) {
152    if (amount[i] == '.') {  153             if (decimal)
154                 break;
155             decimal++;
156             continue;
157         }
158         if (!isdigit(amount[i]))
159             break;
160         if (decimal) {
161             decimal++;
162             if (decimal > 3)
163                 break;
164         }
165     }
166     if (i == 0 || (3 - decimal) + i > 10) {
167         (void) NoteAlert(amounCAUT, NULL);
168         return(FALSE);
169     }
```

The actual work of the application begins with the comments at line 412 (not shown). Each of the dialog boxes has a set of routines which are used to implement its functionality.

The dialog boxes, which take up the rest of the code, accept input from the user, start a transaction, make a call to the requested service and return information to the user. The code in Listing 7-4 shows how errors might be handled. In lines 594-599, for example, if the call to `tpbegin`(3c) fails, a message is sent to `userlog(3c)` and also to the user's screen via the `ParamText` function. In lines 600-605, if the service call fails and the buffer is not NULL, the status line is picked up and returned to the user.

Lines 625-630 show the service request being successfully performed and the requested balance being displayed on the user's screen.

Listing 7-4 bankapp for Macintosh: Error Handling

```
594     if (tpbegin(30, 0) == -1) {
595       (void) userlog("failed to begin transaction");
596         NumToString(tperrno, tpStr);
597         ParamText("\pOpen Account Failed", "\p",
598                     "\ptperrno = ", tpStr);
599     }
600     else if (tpcall("OPEN_ACCT", (char *) *fbfr, 0, (char *) *fbfr,
          &len, 0) == -1) {
601         if((tperrno == TPESVCFAIL) & (*fbfr != NULL) &
602                 ((s = Ffind(*fbfr,STATLIN,0,0)) != 0)) {
603             BlockMove(s, account1[1], (Size) (account1[0] =
                  (unsigned char) strlen(s)));
604             ParamText((unsigned char *) account1, "\p", "\p", "\p");
605         }
606         else {
607             NumToString(tperrno, tpStr);
608             ParamText("\pTransfer Failed", "\p", "\ptperrno = ", tpStr);
609         }
625             Fget(*fbfr, ACCOUNT_ID, 0, (char *) acc_num, 0);
626             s = Ffind(*fbfr, SBALANCE, 0, 0);
627             NumToString(acc_num, tpStr);
628             BlockMove(s, account1[1], (Size) (account1[0] =
                        (unsigned char) strlen(s)));
629             ParamText("\pAccount Number: ", tpStr,
630                    "\pAccount Balance: ", (unsigned char *) account1);
```

Blocking Network Behavior

When an ATMI function is called the Macintosh could block on the network waiting for a reply from the server. This can happen on any call that initiates a network message to the UNIX machine, for example, `tpcall`(3c), `tpinit`(3c), `tpgetrply`(3c) and so on. These functions may take an arbitrarily long time to complete; a good example is `tpcall(3c)`, which may block until the server has completed the processing required.

With the BEA TUXEDO Workstation for Macintosh a blocking operation that cannot be completed immediately is handled as follows. The Macintosh initiates the operation and enters a loop in which it calls `WaitNextEvent` while retrieving only `NULL` events (yielding the processor to another process if necessary). It then checks for the completion of the ATMI function. If the ATMI call is complete the blocking call is completed and the appropriate result is returned to the caller. If the ATMI call is not complete the Macintosh continues to dispatch `WaitNextEvent` messages. For a complete description of this behavior see `AEMsetblockinghook`(3c) in the *BEA TUXEDO Reference Manual*.

Although this mechanism is sufficient for simple applications, it cannot support the complex message dispatch requirements of more advanced applications. For such applications, ATMI includes `AEMsetblockinghook`(3c), which allows the programmer to define a special routine which will be called instead of the default routine.

By using `AEMsetblockinghook`(3c), there is a risk that the application will attempt to issue another ATMI call. Such application behavior is not supported by ATMI calls. `AEMisblocked`(3c) can be called at any time to detect if there is a blocking ATMI call outstanding. Any other ATMI call made while this condition exists will fail and set `tperrno` to `TPEPROTO`.

If an application invokes a blocking operation like `tpcall()` and provides a typed buffer to it as an argument, it is the responsibility of the application to ensure that the buffer is available to ATMI until the operation is completed.

FML functions will continue to work even if there is a blocking call in progress, therefore it is the responsibility of the application to not use FML buffers that are passed in as an argument to an ATMI call that is currently in progress until the ATMI call completes.

Building Client Programs

Any compiler that can read MPW C libraries can be used to compile application programs.

When compiling with MPW, be sure to use the `-model far`, `-m`, and `-mc68020` options.

When using THINK C, use the same compiler options as specified above for library generation.

A Macintosh executable, `TuxShell`(1), is provided to run BEA TUXEDO utilities. `TuxShell`(1) takes as input a script containing utility command lines. For details, see `TuxShell`(1) in the *BEA TUXEDO Reference Manual*.

Runtime

When you run client programs, the `TUXDIR` environment variable must be set properly before `tpinit`(3c) or `tpalloc`(3c) is called. In order to use the BEA TUXEDO catalogs properly, the `locale:C` directory must be below the `{TUXDIR}` directory. You may want to put the `locale:C` directory into the System Folder, since the application can always find that filesystem.

Limitations

The following is a list of limitations that apply to Release 5.0 (or higher) of the BEA TUXEDO system Macintosh libraries.

- The THINK C, version 7.1, compiler is required for building the `TuxShell`(1) program.
- Multiplexed network connections are not available; each client process requires a separate network connection.

8 BEA TUXEDO Workstation for OpenVMS

What This Chapter Is About

This chapter describes the installation and use of the BEA TUXEDO Workstation on a DEC Alpha configured with OpenVMS 6.2. This platform offers developers the ability to write application clients using OpenVMS.

The major sections in this chapter cover:

- Software prerequisites
- BEA TUXEDO system client programs and OpenVMS

Prerequisites

This section lists the hardware and software prerequisites.

Hardware

- The BEA TUXEDO OpenVMS /WS can be installed successfully on any DEC Alpha platform that is supported by OpenVMS 6.2.
- For an acceptable level of performance, the machine should have 64MB of RAM.
- The machine on which the BEA TUXEDO Workstation for OpenVMS is installed runs as a remote machine to a UNIX server.

Software

- OpenVMS 6.2 must be installed and configured before the BEA TUXEDO Workstation is installed.
- If you intend to install the BEA TUXEDO online documentation, it is your responsibility to make sure a Web browser is available. The online documentation has been tested with Netscape 2.0, 3.0, and MicroSoft Internet Explorer 3.0.
- Digital TXP/IP Services V4.1 for OpenVMS (UCX 4.1)
- DEC C and/or DEC COBOL compilers

The BEA TUXEDO server machine must have the BEA TUXEDO system and the native-side BEA TUXEDO Workstation installed.

Building and Running a Sample Client Program

This section covers items specific to building and running a BEA TUXEDO Workstation for OpenVM client program. It is intended to illustrate material presented in the *BEA TUXEDO Programmer's Guide* and in the *BEA TUXEDO COBOL Guide*. Our assumption is that readers of this section either have experience in writing OpenVMS programs or have access to tutorial material on that subject. Our presentation is limited to showing you a very simple client, `simpcl` (from a sample application known as `simpapp`). In the code you will see BEA TUXEDO ATMI calls used.

Writing Client Programs

The ATMI calls used in OpenVMS client programs are the same as those described in the chapter entitled "Writing Client Programs" in the *BEA TUXEDO Programmer's Guide*.

Using simpapp as an Example

After the BEA TUXEDO Workstation software is installed, among the files in the directory `$TUXDIR/apps/simpapp` are those shown in Listing 8-1. These are the files needed to produce the `simpapp` client for OpenVMS.

Listing 8-1 simpapp Files for OpenVMS

```
README
simpapp.mk
simpcl.c
simpserv.c
ubbmp
ubbsimple
ubbws
```

You may find you have additional files; if so, they are extraneous files appropriate for other platforms.

Listing 8-2 shows the source code for `simpcl.c`.

Listing 8-2 simpapp for OpenVMS: Client Program

```
/* #ident       "@(#)apps:simpapp/simpcl.c      60.3" */
#include
#include "atmi.h"               /* TUXEDO  Header File */

#if defined(__STDC__) || defined(__cplusplus)
main(int argc, char *argv[])
#else
main(argc, argv)
int argc;
char *argv[];
#endif
{
        char *sendbuf, *rcvbuf;
        long sendlen, rcvlen;
        int ret;
        if(argc != 2) {
                (void) fprintf(stderr, "Usage: simpcl string\n");
                exit(1);
        }
        /* Attach to System/T as a Client Process */
        if (tpinit((TPINIT *) NULL) == -1) {
                (void) fprintf(stderr, "Tpinit failed\n");
                exit(1);
        }

        sendlen = strlen(argv[1]);
        /* Allocate STRING buffers for the request and the reply */
        if((sendbuf = (char *) tpalloc("STRING", NULL, sendlen+1)) == NULL) {
                (void) fprintf(stderr,"Error allocating send buffer\n");
                tpterm();
                exit(1);
        }
        if((rcvbuf = (char *) tpalloc("STRING", NULL, sendlen+1)) == NULL) {
                (void) fprintf(stderr,"Error allocating receive buffer\n");
                tpfree(sendbuf);
                tpterm();
                exit(1);
        }
        (void) strcpy(sendbuf, argv[1]);
```

```
        /* Request the service TOUPPER, waiting for a reply */
        ret = tpcall("TOUPPER", (char *)sendbuf, 0, (char **)&rcvbuf, &rcvlen,\
            (long)0);
        if(ret == -1) {
               (void) fprintf(stderr, "Can't send request to service TOUPPER\n");
                (void) fprintf(stderr, "Tperrno = %d\n", tperrno);
                tpfree(sendbuf);                tpfree(rcvbuf);
                tpterm();
                exit(1);
        }
        (void) fprintf(stdout, "Returned string is: %s\n", rcvbuf);
        /* Free Buffers & Detach from System/T */
        tpfree(sendbuf);
        tpfree(rcvbuf);
        tpterm();
        return(0);
}
```

There are three things to observe in `simpcl`; these things illustrate calls you will use a lot as you move on to code client programs for your own applications.

`simpcl` calls `tpinit()` to attach to BEA TUXEDO as a client. In this case, the call is made without allocating and freeing a `TPINIT` buffer. This practice is allowable if you are unconcerned about the client authentication process; in this simple example we are not concerned about it. For further details about this ATMI call, see the `tpinit`(3c) reference page.

The next items of interest are the two calls to `tpalloc` to get space for send and receive buffers. A full description of `tpalloc`(3c) can be found on the `tpalloc`(3c) reference page. Note that the space is freed at the end of the program by calls to `tpfree`(3c).

The real work of the program is done in the call to `tpcall`(3c). `tpcall` is a BEA TUXEDO system ATMI function that waits while the server processes its request and sends back the response. An alternative, that may be appropriate if your request takes a while to process, is to use `tpacall`(3c) and `tpgetreply`(3c). `tpacall` issues a service request and checks back later for the reply. More information about `tpcall` can be found on the `tpcall`(3c) reference page.

In those few lines, you have all the components of a real client program.

Building BEA TUXEDO Workstation Client Programs

Any compiler that can read DEC C libraries can be used to compile application programs.

Use the CFLAGS environment variable to pass any additional application flags to the DEC C compiler.

Use the TMLKFLAGS environment variable to pass any additional application flags to the linker.

Use the buildclient(1) command to build /WS client programs. You should first set the environment by executing the file tux_env.com. The two commands look like this:

```
$ @tux-env.com
$ buildclient -w -o wsimpcl.exe -f simpcl.c
```

Setting the Environment for Running Client Programs

When you run client programs, the TUXDIR environment variable must be set properly before tpinit(3c) or tpalloc(3c) is called.

A file of environment variables, tux_env.com, is delivered with BEA TUXEDO Workstation for OpenVMS. To run the file, use the procedure shown in Listing 8-3.

Listing 8-3 Setting the Environment and Running wsimpcl

```
$ set default
$ @[]tux_env.com
$ set default[.appx.simpapp]
$ WSNADDR=<host:port>
$ WSTYPE:=VMS
$ wsimpcl:='f$environment("default")'wsimpcl.exe
```

Replace the values shown (in Listing 8-3) in angle brackets (< >) with values from your location. For TUXDIR, substitute the name of the root directory where BEA TUXEDO OpenVMS /WS is installed. For WSNADDR, enter the TCP/IP address for the workstation listener process on your BEA TUXEDO system server. Include the port number if necessary. Check with your system administrator if you need help. For information on the correct format for network addresses, see WSL(5).

Limitations

`wmio`(1) is not supported.

`wtmconfig`(1) is not supported.

9 Bringing Up Bankapp on Workstations

What This Chapter Is About

This chapter describes the steps to follow in bringing up bankapp, the BEA TUXEDO system sample application, on a UNIX or MS-DOS workstation.

Characteristics of a Workstation Application

Client processes are moved off the native site. Listener process (WSL) runs with a well-known network address and starts surrogate workstation handlers (WSH) as needed. Servers run on one or more UNIX machines within the BEA TUXEDO system administrative domain.

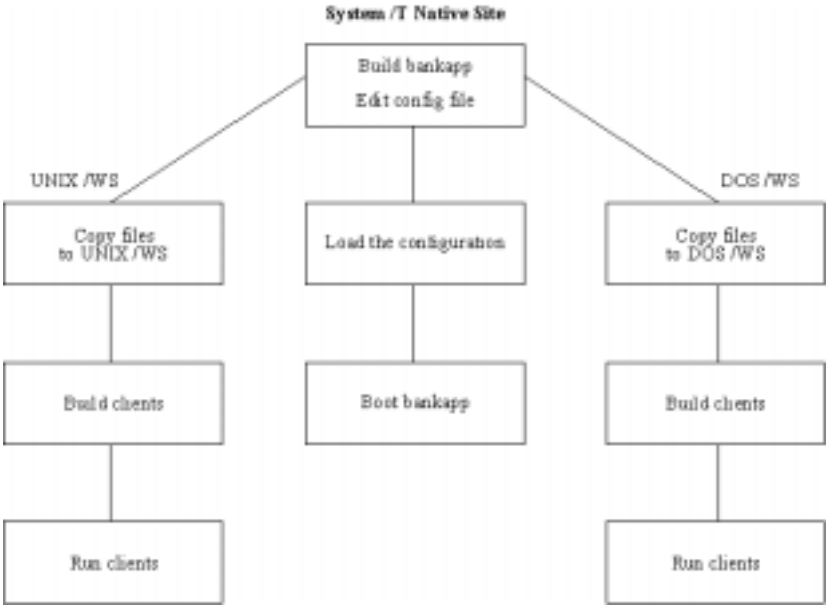
Overview of the Enhanced bankapp

- Existing `bankapp` client programs, consisting of several data entry masks and the `audit.c` client program, are available to run on a UNIX workstation.
- New client programs, or new versions of existing ones, are provided to run on MS-DOS workstations.
- Existing servers are available to run on the BEA TUXEDO system nodes in either SHM or MP mode.

The Process Diagrammed

Figure 9-1 shows the steps in the process of bringing up `bankapp` on workstations, and also provides an outline of the subjects in this chapter.

Figure 9-1 Steps in bringing up bankapp



Changes on the Native Site

Install and build the bankapp software on the native site. The procedure for doing this is described in the *BEA TUXEDO Application Development Guide* and in the README file found in $TUXDIR/apps/bankapp on the master machine where your BEA TUXEDO system software is installed.

New Configuration File Parameters

You need to edit the configuration file you plan to use (either ubbshm or ubbmp) to specify the workstation listener server, WSL, in the GROUPS and SERVERS sections and to specify MAXWSCLIENTS in the MACHINES section. When you edit the GROUPS section, put the entry for WSGRP ahead of the DEFAULT line or move the specifications for TMSNAME and TMSCOUNT to the server groups that use them; they should not be assigned to WSGRP. The new specifications should be in the following format.

```
*MACHINES
 DEFAULT:   MAXWSCLIENTS=50

 #
 *GROUPS

 WSGRP      GRPNO=<next available group #> LMID=SITE1
 #
 *SERVERS

 WSL        SRVGRP=WSGRP            SRVID=1
            CLOPT="-A – -n 0x0002ffffaaaaaaaa -d /dev/tcp -m 1 -M 5 -x 10"
```

Also, remember to increase the MAXACCESSERS parameter in the RESOURCES or MACHINES section to cover the new workstation clients.

Load and Boot the Configuration

At some point, before you can start using /WS clients, you need to run tmloadcf to load the configuration file into its binary form and tmboot to start the application. These commands do not have to be run immediately; there is work to be done in getting the bankapp clients installed on your workstations and getting them built. However, the application must be running on the BEA TUXEDO system native site when you attempt to join the application from a workstation. The steps for loading and booting bankapp on the native site are part of the overall procedure documented in the *BEA TUXEDO Application Development Guide*.

bankapp on a UNIX Workstation

This section covers the installation of bankapp client programs on a UNIX workstation.

Install the Files

The source files shown in Listing 9-1 are the bankapp files that you need to copy from TUXDIR/apps/bankapp to the client workstation.

Listing 9-1 Files for UNIX /WS clients

```
BALANCE.m   Mask for balance inquiry data entry.
CBALANCE.m  Mask for confirmation of a balance inquiry.
CCLOSE.m    Mask for confirmation of an account closing.
CDEPOSIT.m  Mask for confirmation of an deposit.
CLOSE.m     Mask for account closing data entry.
COPEN.m     Mask for confirmation of an account closing.
CTRANSFER.m Mask for confirmation of a transfer.
CWITHDRAW.m Mask for confirmation of a withdrawal.
DEPOSIT.m   Mask for deposit data entry.
HELP.m      Mask that explains mio keystrokes
MENU.m      Initial mask that offer a ring menu
             to choose deposit, withdrawal, transfer, balance inquiry,
             open account, or close account data entry screens.
OPEN.m      Mask for open account data entry.
TRANSFER.m  Mask for transfer data entry.
WITHDRAW.m  Mask for withdrawal data entry.
aud.v       FML view used to define structure passed between
             audit client the BAL server.
audit.c     Source code for audit client program.
bank.flds   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
             one C program within the application.
bankvar     Contains environment variable and sets those contained
             in ENVFILE.
credit.flds Field table file containing credit card fields used by
             masks and servers.
driver.sh   Drives the application by piping FML buffers
```

```
              with transaction requests through wud(1).
envfile.sh    envfile ENVFILE Creates ENVFILE for use by tmloadcf.
gendata.c     Generates wud(1) readable transaction request to
              add ten branches, thirty tellers and two hundred accounts.
gentran.c     Generates wud(1) readable transaction requests
              from among the DEPOSIT, WITHDRAWAL, INQUIRY, and TRANSFER services.
populate.sh   Populates the database by piping FML buffers generated
              by gendata through wud(1).
run.sh        Invokes mio with MENU mask.
wsbankapp.mk  Application makefile for UNIX workstations.
```

Set bankapp Variables

Edit a file `wsenv` to include the following variables (with appropriate settings) needed on the workstation side:

```
WSNADDR=<WSL advertised address(es)>
WSDEVICE=<device name of /WS network provider>
WSTYPE=<type of /WS machine>
```

Edit `bankvar` to point to the correct `TUXDIR` and `APPDIR` on the UNIX workstation. Also add the line

```
WSENVFILE={$TUXDIR}/wsenv;export WSENVFILE
```

Execute `bankvar` with the command:

```
. ./bankvar
```

Build the bankapp Clients

Run

```
make -f wsbankapp.mk
```

to build the client programs.

Run the bankapp UNIX Workstation Clients

Edit the shell script `run` to change the line.

```
mio -i MENU
```

to

```
wmio -i MENU
```

Then execute `run`.

bankapp on an MS-DOS Workstation

This section covers the installation of `bankapp` client programs on an MS-DOS workstation.

Install the Files

The source files for BEA TUXEDO Workstation clients for MS-DOS are named in the file `ws/dosfiles` on the native site. They reside in the directory `$TUXDIR\apps\ws` after you have completed the installation procedure. The files are listed in Listing 9-2.

Listing 9-2 Files for MS-DOS Workstation Clients

```
BANKAPP.C    DOS Client software
BANKAPP.H    Header file for DOS client
BANKFLDS     FML field definitions
FILES        List of files
MSC.MAK      Microsoft C makefile
```

Build the bankapp Clients

The client programs for `bankapp` are not built when you run `gentux`, which you use to build your /WS MS-DOS software. The makefiles assume that the Large memory model libraries have been installed in `TUXDIR\lib` for the MS-DOS client, `bankapp.exe`. The makefiles also assume that the network provider is Novell sockets (libraries `llibsock.lib` and `wlibsock.lib` are referenced). These assumptions can be changed by modifying the makefiles described below. The environment must be set up as described in "Building Client Programs" in Chapter 5.

When building with the Microsoft compiler, the `msc.mak` makefile must be used. To run the MS-DOS client, the `HELVB.FON` file is needed (it is normally located in `\c600\src\samples`, `\c700\lib`, or `\msvc\lib\font`).

```
nmake -f msc.mak bankapp.exe
```

Run the bankapp DOS Workstation Clients

To run the clients, set up the environment variables as described in "Running BEA TUXEDO System Clients on a Workstation" in Chapter 4. If `bankapp` has not already been booted on the native site, it must be done before you attempt to run Workstation clients. Assuming `bankapp` is running, execute `bankapp` to run MS-DOS clients, or `win bankappw` to run Windows clients.