BEA Tuxedo®

Using the BEA Tuxedo ATMI Workstation Component

Version 10.0
Document Released: September 28, 2007

Contents

4. Bringing Up bankapp on Workstations

CHAPTER 1

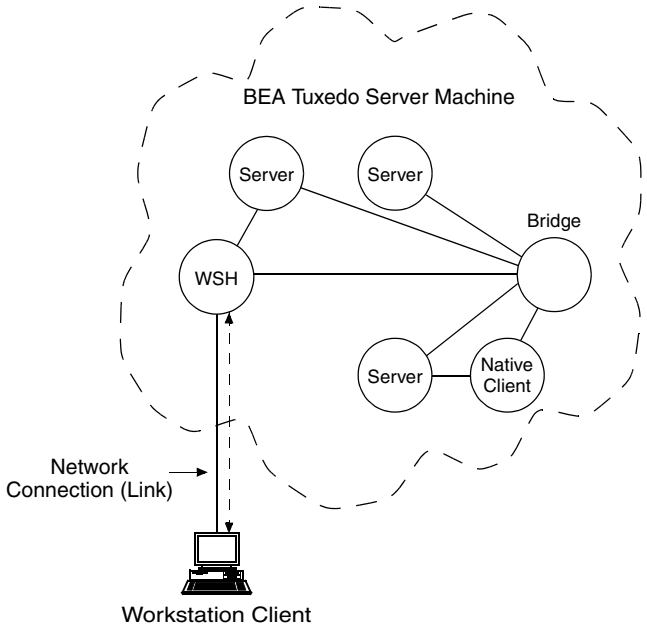
Workstation Overview

The following sections provide a brief overview of the BEA Tuxedo ATMI Workstation component:

What Is the Workstation Component?

The Workstation component of the BEA Tuxedo system allows application clients to reside on a machine that does not have a full server-side installation, that is, a machine that does not support any administration or application servers. As shown in the following figure, all communication between a *Workstation client*—an application client running on a Workstation component—and the server application takes place over the network.

Figure 1-1 BEA Tuxedo ATMI Workstation Component Operation



Note: A Workstation client communicates with a server application through a workstation handler (WSH) process.

A Workstation client, whether run on a Windows or UNIX system, has access to most of the client ATMI, although a Workstation client does not have all the access privileges available to a native client (that is, a client running on the same machine on which the server program is running). However, both types of clients can do the following:

- Send and receive messages
- Begin, end, and commit transactions
- Send and receive unsolicited messages
- Meet application security requirements
- Communicate information about remote clients through the `tmadmin()` command (for details, see `tmadmin(1)` in *BEA Tuxedo Command Reference*)

Limitations of Workstation Clients

Workstation clients do not have access to all the functionality available to native clients. For example, unlike a native client, a Workstation client cannot join an application as `tpsysadm`, which means that the client cannot subsequently subscribe to an event that issues a service call.

See Also

- "BEA Tuxedo Workstation Servers" in *Introducing BEA Tuxedo ATMI*
- "About Workstation Clients" in *Setting Up a BEA Tuxedo Application*

Workstation Administration

To integrate a Workstation client into a BEA Tuxedo application, you must define any required and desired parameters for that client in the application configuration file. For details, see "Setting Up Workstation Clients" in *Setting Up a BEA Tuxedo Application.*

See Also

- `buildwsh(1)` in *BEA Tuxedo Command Reference*
- `WSL(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*

CHAPTER 2

Using the Workstation Component

The following sections describe using the BEA Tuxedo ATMI Workstation component on both Windows and UNIX systems:

- Writing Client Programs
- Using BEA Tuxedo System-Supplied Clients
- Running BEA Tuxedo System Clients on a Workstation
- How a Multithreaded or Multicontexted Workstation Client Joins an Application

Writing Client Programs

You can develop client programs targeted for workstations in the same way that you develop client programs within the BEA Tuxedo system administrative domain (that is, native clients). With a few exceptions, all ATMI and FML functions available to the native client are also available to the Workstation client.

Note: `tpadmcall()` is an example of an ATMI function that is available to the native client but not to the Workstation client.

Interoperability Restrictions for Workstation Clients

Interoperability between BEA Tuxedo release 7.1 or later Workstation clients and applications based on pre-7.1 releases of the BEA Tuxedo system is supported in any of the following situations:

- The client is neither multithreaded nor multicontexted.
- The client is multicontexted.
- The client is multithreaded and each thread is in a different context.

A BEA Tuxedo release 7.1 or later Workstation client with multiple threads in a single context cannot interoperate with a pre-7.1 release of the BEA Tuxedo system.

Building Client Programs

You compile and link-edit Workstation client programs using 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 to indicate the client should be built using the workstation libraries. Otherwise, on a native node, where both native and workstation libraries are present, the default is to use the native libraries. In this case, using 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` option.

The following listing shows an example of the `buildclient(1)` command line on a native node.

Listing 2-1 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 output file. Input files are specified with the `-f` *`firstfiles`* option to indicate that they are link-edited before system libraries. As indicated in the example, you must define the `TUXDIR` environment variable to ensure that the `buildclient` command can locate system libraries. `CC` defaults to `cc` but can be set to another compiler, as shown in the example.

See Also

- “Writing Clients” in *Programming a BEA Tuxedo ATMI Application Using C* and *Programming a BEA Tuxedo ATMI Application Using COBOL*
- “COBOL Language Bindings for the Workstation Component” in *Programming a BEA Tuxedo ATMI Application Using COBOL*

- "Writing Security Code So Client Programs Can Join the ATMI Application" in *Using Security in CORBA Applications*
- `buildclient(1)` in *BEA Tuxedo Command Reference*

Using BEA Tuxedo System-Supplied Clients

`wud` and `wud32` are BEA Tuxedo system-supplied driver programs provided for workstations. These driver programs are based on the standard BEA Tuxedo client programs, `ud` and `ud32`, that have been built using the workstation libraries.

Use `wud(1)` to send `FML` buffers to BEA Tuxedo system servers. Use `wud32` with fielded FML32 buffers of type `FBFR32`.

Using wud in a Security Application

If `wud` is run in a security application, it requires an application password to access the application. If standard input is from a terminal, `wud` prompts the user for an application password. If you are running the client program from a script, which is a common occurrence with `wud`, the password is retrieved from the environment variable `APP_PW`. If this environment variable is not specified and an application password is required, then `wud` fails.

Do not confuse the `APP_PW` environment variable with the similar configuration file parameter, `SECURITY`, for which the value `APP_PW` enables the security feature.

See Also

- `ud, wud(1)` in *BEA Tuxedo Command Reference*

Running BEA Tuxedo System Clients on a Workstation

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

Verifying the Directory Structure on Workstation Clients

The following table describes the directory structure on a Workstation client after you have installed the Workstation component of the BEA Tuxedo system.

Windows Directory	UNIX Directory	Description
%APPDIR%	$APPDIR	Contains the client executables. These executables are commonly kept in the directory from which the application is run.
%TUXDIR%\bin	$TUXDIR/bin	Contains BEA Tuxedo system commands and system clients such as wud.
%TUXDIR%\cobinclude	$TUXDIR/cobinclude	Contains copylib entries for use in COBOL programs.
%TUXDIR%\include	$TUXDIR/include	Contains BEA Tuxedo system header files such as atmi.h.
%TUXDIR%\lib	$TUXDIR/lib	Contains run-time libraries.
%TUXDIR%\locale\C	$TUXDIR/locale/C	Contains message catalogs for the default locale (U.S. English).
%TUXDIR%\samples	$TUXDIR/samples	Contains several subdirectories with sample applications.

Setting Environment Variables

Workstation clients make use of several environment variables. The following table shows the environment variables that are checked by tpinit(3c) or TPINITIALIZE(3cbl) when the Workstation client attempts to join the application. For details on setting these environment variables, see "Defining Workstation Clients" in *Setting Up a BEA Tuxedo Application.*

Environment Variable	Description
TPMBENC	Specifies the code-set encoding name that the workstation machine running BEA Tuxedo 8.1 or later includes in an allocated MBSTRING typed buffer. When a Workstation client allocates and sends an MBSTRING buffer, the code-set encoding name defined in TPMBENC is automatically added as an attribute to the buffer and sent with the buffer data to the destination server process. When the workstation machine receives an MBSTRING buffer, and assuming another environment variable named TPMBACONV is set, the code-set encoding name defined in TPMBENC is automatically compared to the code-set encoding name in the received buffer; if the names are *not* the same, the MBSTRING buffer data is automatically converted to the encoding defined in TPMBENC before being delivered to the Workstation client. TPMBENC has no default value. For a Workstation client using MBSTRING typed buffers, TPMBENC must be defined on the workstation machine. **Note:** TPMBENC is used in a similar way for FLD_MBSTRING fields in an FML32 typed buffer.
TPMBACONV	Specifies whether the workstation machine running BEA Tuxedo 8.1 or later automatically converts the data in a received MBSTRING buffer to the encoding defined in TPMBENC. By default, the automatic conversion is turned off, meaning that the data in the received MBSTRING buffer is delivered to the Workstation client *as is*—no encoding conversion. Setting TPMBACONV to any non-NULL value, say Y (yes), turns on the automatic conversion. **Note:** TPMBACONV is used in a similar way for FLD_MBSTRING fields in an FML32 typed buffer.

Environment Variable	Description
URLENTITYCACHING	Specifies whether the workstation machine running BEA Tuxedo 8.1 or later caches Document Type Definition (DTD), XML schema, and entity files; specifically, whether the Apache Xerces-C++ parser running on the Workstation client caches the DTD and XML schema files when validation is required, or caches external entity files called out in the DTD. By default, the caching is turned on (Y). Setting URLENTITYCACHING to N (no) turns off the caching.
URLENTITYCACHEDIR	Applies only if URLENTITYCACHING=Y (yes) or is not set; for details, see the description of URLENTITYCACHING in this table. Specifies the directory in which the workstation machine running BEA Tuxedo 8.1 or later caches DTD, schema, and entity files; specifically, where the Apache Xerces-C++ parser running on the Workstation client caches the DTD, XML schema, and entity files. The URLENTITYCACHEDIR variable specifies the absolute pathname for the cached files. If URLENTITYCACHEDIR is not specified, the default directory becomes URLEntityCachedir, which will be created in the current working directory of the Workstation client process provided that the appropriate write permissions are set.
WSINTOPPRE71	Specifies whether the workstation machine running BEA Tuxedo 7.1 or later is allowed to interoperate with pre-release 7.1 BEA Tuxedo applications. Setting the variable to Y (WSINTOPPRE71=Y) allows interoperability.
WSBUFFERS	The number of packets per application.
WSDEVICE	Device name to be used to access the network. This variable is only required when the BEA Tuxedo system is using the TLI networking interface.
WSENVFILE	Name of a file containing environment variable settings to be set in the client's environment.

Environment Variable	Description
WSFADDR	The network address used by the Workstation client when connecting to the workstation listener or workstation handler. This variable, along with the WSFRANGE variable, determines the range of TCP/IP ports to which a Workstation client will attempt to bind before making an outbound connection. This address must be a TCP/IP address.
WSFRANGE	The range of TCP/IP ports to which a Workstation client process attempts to bind before making an outbound connection. The WSFADDR parameter specifies the base address of the range. The default is 1.
WSNADDR	The network address of the workstation listener (WSL) process through which clients gain access to the application. Use the value specified in the application configuration file for the workstation listener to be called. If the value begins with the characters 0x, the system interprets it as a string of hexadecimal digits; otherwise, the system interprets it as ASCII characters.
WSRPLYMAX	Maximum amount of core memory that ATMI functions use for buffering application replies before they are dumped to disk. Used by tpinit(3c) and TPINITIALIZE(3cbl). Replies obtained using tpgetrply(3c), TPGETRPLY(3cbl), and unsolicited messages are buffered in this area. When this area is filled with one or more messages, the overflow is written to a disk file. The system default limit is 256,000 bytes. Whether you use WSRPLYMAX to set a lower limit depends on the amount of available memory on your machine. Writing replies to disk causes a substantial reduction in performance.
WSTYPE	Workstation type. Used within tpinit(3c) and TPINITIALIZE(3cbl) when invoked by a Workstation client to negotiate encode/decode responsibilities with the native site. If you do not specify WSTYPE, the system performs encoding, even if it is also unspecified on the native site.You must explicitly specify the same WSTYPE value for both sites to ensure that the encode/decode feature is turned off.

Other environment variables may be needed by Workstation clients on a UNIX workstation depending on the BEA Tuxedo system features being used. Reference page `compilation(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference* explains which variables are needed under what circumstances.

Creating an Environment File

If you have created an environment file, it is read when `tpinit(3c)` or `TPINITIALIZE(3cbl)` is called. The following listing shows a sample file that could be used for two different applications.

Listing 2-2 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

When you call the tuxreadenv(3c) function, it reads the environment file and adds the environment variables to the environment for the entire process, independent of platform. These variables are available using tuxgetenv(3c) and can be reset using tuxputenv(3c).

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

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

- %TUXDIR%\TUXEDO.ENV (Windows)
- $TUXDIR/TUXEDO.env (UNIX)

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 under the following conditions:

- A memory failure
- A non-null filename does not exist
- A non-null label does not exist

Each time `tpinit(3c)`is called (either explicitly or implicitly by calling another ATMI function), `tuxreadenv(3c)` 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 filename 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 the following:

- All variables from the specified `WSENVFILE` file for the specified `WSAPP` label (or the defaults if not specified)
- The environment variables in the operating system environment

See Also

- `tpinit(3c)` in *BEA Tuxedo ATMI C Function Reference*
- `tuxreadenv(3c)` in *BEA Tuxedo ATMI C Function Reference*

How a Multithreaded or Multicontexted Workstation Client Joins an Application

To join a BEA Tuxedo application, a multithreaded Workstation client must always call `tpinit()` with the `TPMULTICONTEXTS` flag set, even if the client is running in single-context mode.

See Also

- `tpinit(3c)` in *BEA Tuxedo ATMI C Function Reference*

CHAPTER 3

Using Workstation on a Windows System

The following sections describe additional information about using the BEA Tuxedo ATMI Workstation component on a Windows XP or Windows Server 2003 system:

- Benefits of Using Workstation on a Windows System
- Software Prerequisites
- Writing Client Programs
- How a Multithreaded or Multicontexted Workstation Client Joins an Application

Benefits of Using Workstation on a Windows System

The Windows instantiation of the Workstation client offers significant benefits to application developers:

- 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.

Software Prerequisites

The software prerequisites for running the Workstation component on a Windows system are as follows:

- Workstation for Windows runs under the Windows XP or Windows Server 2003 operating system.
- In Windows, the native TCP/IP stack is used.
- In Windows, while using TCP/IP, any Windows Sockets Compliant TCP/IP stack can be used.
- The server machine must have the BEA Tuxedo system and the native-side BEA Tuxedo Workstation installed.

Writing Client Programs

You can develop client programs targeted for Windows workstations in the same way that you would develop native client programs within the BEA Tuxedo system administrative domain. All the ATMI functions are available.

Interoperability Restrictions for Workstation Clients

Interoperability between BEA Tuxedo release 7.1 and later Workstation clients and applications based on pre-7.1 releases of the BEA Tuxedo system is supported in any of the following situations:

- The client is neither multithreaded nor multicontexted.
- The client is multicontexted.
- The client is multithreaded and each thread is in a different context.

A BEA Tuxedo release 7.1 or later Workstation client with multiple threads in a single context cannot interoperate with a pre-7.1 release of the BEA Tuxedo system.

Building Client Programs

To compile client programs written in C, you can use any compiler that can read Microsoft C import libraries. To compile COBOL source programs that call the ATMI, use the `LITLINK`

option of the COBOL compiler. For details, see "COBOL Language Bindings for the Workstation Component" in *Programming a BEA Tuxedo ATMI Application Using COBOL*.

Use `buildclient(1)` with the `-w` flag to link-edit your client programs.

You can also build BEA Tuxedo clients without using the `buildclient(1)` utility. If you are using Microsoft Visual C++ projects, use the following settings:

- Set the Preprocessor option to `-DWIN32`.
- Add `WTUXWS32.LIB` `MSVCRT.LIB` to the input libraries for the linker option.

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

Building GUI ATMI Clients

The `sample/atmi/ws` directory contains several different windows platform `.mak` files for creating GUI atmi clients. For an example of how these files may be used, see Tutorial for bankapp, a Full C Application in *Tutorials for Developing BEA Tuxedo ATMI Applications.*

Run Time

When you run client programs, your `PATH` must includo`TUXDIR%\bin`.

Limitations

The BEA Tuxedo libraries (DLLs) prior to BEA Tuxedo release 7.1 are not thread-safe. For applications written using the pre-release 7.1 DLLs, threads should not be used; otherwise, threaded access is serialized through all BEA Tuxedo calls (such as ATMI, FML, `userlog()`, and so on).

See Also

- "Writing Clients" in *Programming a BEA Tuxedo ATMI Application Using C* or *Programming a BEA Tuxedo ATMI Application Using COBOL*
- "COBOL Language Bindings for the Workstation Component" in *Programming a BEA Tuxedo ATMI Application Using COBOL*
- "Writing Security Code So Client Programs Can Join the ATMI Application" in *Using Security in CORBA Applications*
- `buildclient(1)` in *BEA Tuxedo Command Reference*

How a Multithreaded or Multicontexted Workstation Client Joins an Application

To join a BEA Tuxedo application, a multithreaded Workstation client must always call `tpinit(3c)` with the `TPMULTICONTEXTS` flag set, even if the client is running in single-context mode.

See Also

- `tpinit(3c)` in *BEA Tuxedo ATMI C Function Reference*

CHAPTER 4

Bringing Up bankapp on Workstations

The following sections describe the procedure for bringing up `bankapp`, the BEA Tuxedo system sample application, on a Windows or UNIX workstation:

- Characteristics of a Workstation Application
- Bringing Up bankapp on a Workstation Client
- Changes on the Native Site
- Setting the bankapp Variables
- Building the bankapp Client
- Running the bankapp Client

Characteristics of a Workstation Application

In a workstation application, client processes are moved off the native site. The listener process (WSL) runs with a well-known network address and starts surrogate workstation handlers (WSH) as needed. Servers run on one or more machines within the BEA Tuxedo administrative domain.

Existing servers are available to run on the BEA Tuxedo system nodes in either single processor (SHM) or multiprocessor (MP) mode.

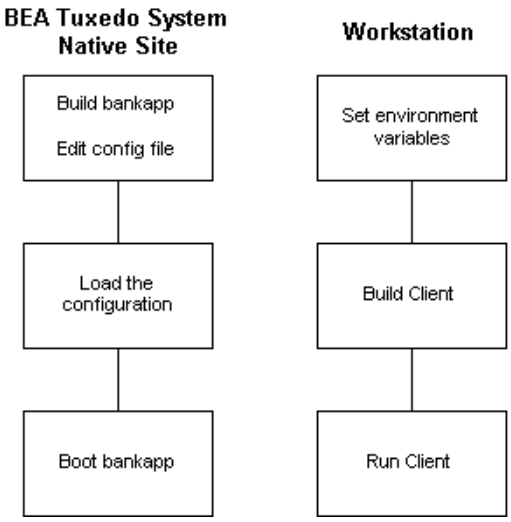
On Workstations, the sample applications are located in the following directories:

- `%TUXDIR%\samples\atmi\bankapp` (Windows)
- `$TUXDIR/samples/atmi/bankapp` (UNIX)

Bringing Up bankapp on a Workstation Client

The following illustration shows the steps in bringing up bankapp on a Workstation client.

Figure 4-1 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 "Tutorial for bankapp, a Full C Application" in *Tutorials for Developing BEA Tuxedo ATMI Applications* and in the following README files on the master machine where your BEA Tuxedo system software is installed:

- %TUXDIR%\samples\atmi\bankapp\README.nt (Windows)
- $TUXDIR/samples/atmi/bankapp/README (UNIX)

Editing the Configuration File

You need to edit the configuration file you plan to use (either ubbshm or ubbmp) to specify the workstation listener (WSL) as a server 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 //machine:port -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.

Loading and Booting the Configuration

Before you can start using a Workstation client, you need to run tmloadcf(1) to load the configuration file into its binary form and tmboot(1) 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 "Tutorial for bankapp, a Full C Application" in *Tutorials for Developing BEA Tuxedo ATMI Applications*.

See Also

- tmloadcf(1) in *BEA Tuxedo Command Reference*
- WSL(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*
- "Tutorial for bankapp, a Full C Application" in *Tutorials for Developing BEA Tuxedo ATMI Applications*

- "Defining a Workstation Listener (WSL) as a Server" in *Setting Up a BEA Tuxedo Application*

Setting the bankapp Variables

To set your environment to run `bankapp`, complete the following procedure on the Workstation client.

1. Set the following environment variable:

   ```
   WSNADDR=<WSL advertised address(es)>
   WSTYPE=<type of Workstation machine>
   ```

2. Include `%TUXDIR%\bin` (Windows) or `$TUXDIR/bin` (UNIX) in your `PATH`.

3. Verify that your environment is set appropriately to run the C compiler.

Building the bankapp Client

To build a client program, enter the following commands:

```
mkfldhdr bankflds
buildclient -w -o bankclt -f bankclt.c
```

Running the bankapp Client

To run the `bankapp` client on the Workstation, complete the following procedure.

1. Verify that the value of `WSNADDR` on the Workstation client matches the value of the `CLOPT -n` option for the WSL in the `SERVERS` section of the configuration file on the native site.

2. If `bankapp` has not been booted on the native site, make sure it has been booted before you attempt to run a Workstation client.

3. Execute `bankclt` to run the Workstation client.