BEA WebLogic Enterprise

Commands, System Processes, and MIB Reference

WebLogic Enterprise 5.1
Document Edition 5.1
May 2000

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Commands, Processes, and MIB Reference

Document Edition	Date	Software Version
5.1	May 2000	BEA WebLogic Enterprise 5.1

Contents

About This Document

1. Commands Reference

2. Server Process and File Format Reference

3. MIB Reference

About This Document

This document describes the commands used to build and manage BEA WebLogic Enterprise™ (WLE) applications and Enterprise JavaBeans (EJB), the server processes used by WebLogic Enterprise applications, and the BEA Tuxedo® management information base (MIB) classes that were added or enhanced to support WebLogic Enterprise applications.

This document covers the following topics:

- Chapter 1, “Commands Reference,” describes the commands used to build and manage WebLogic Enterprise CORBA applications and WebLogic Enterprise EJBs.
- Chapter 2, “Server Process and File Format Reference,” describes the server processes and file formats used by the WebLogic Enterprise system.
- Chapter 3, “MIB Reference,” describes the BEA Tuxedo management information bases (MIBS) that have been added or enhanced for the WebLogic Enterprise product.

What You Need to Know

This document is intended for programmers who are interested in creating secure, scalable, transaction-based server applications. It assumes you are knowledgeable with CORBA, Enterprise JavaBeans, and the C++ and Java programming languages.

e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the “e-docs” Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Enterprise documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have Adobe Acrobat Reader installed, you can dowload it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA Tuxedo, distributed object computing, transaction processing, C++ programming, and Java programming, see the *WebLogic Enterprise Bibliography* in the WebLogic Enterprise online documentation.

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.1 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

. . .	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Commands Reference

The WebLogic Enterprise system provides the following commands to build and manage WebLogic Enterprise CORBA applications and WebLogic Enterprise EJBs:

- buildjavaserver
- buildobjclient
- buildobjserver
- buildtms
- buildXAJS
- ejbc
- genicf
- idl
- idltojava
- idl2ir
- ir2idl
- irdel
- ISL
- m3idltojava
- tmadmin
- tmboot
- tmconfig

- tmloadcf
- tmshutdown
- tmunloadcf
- tpgrpadd
- tpgrpdel
- tpgrpmod
- tpusradd
- tpusrdel
- tpusrmod
- weblogic.rmc

This topic describes these commands.

buildjavaserver

Synopsis Constructs a Java WLE server application `jar` file.

Syntax `buildjavaserver [-s` *`searchpath`*`] input_file`

Description Once the class files that make up a server application have been created and specified, along with interface activation and transaction policies, in the Server Description File, you use the `buildjavaserver` command to create the `jar` file. The `jar` file contains all the server application class files and a server descriptor. The server descriptor is a serialized Java object that contains:

- Information about all the servant classes implemented by the server application
- Activation and transaction policies for all the interfaces that have been defined in the application's OMG IDL file
- The name of the Server object, which initializes and stops the server application and performs object housekeeping

Options

`-s`
: Specifies a path to be used by the `buildjavaserver` command to locate the classes and packages needed for building the `jar` file. If you do not specify this option, the `buildjavaserver` command uses the class path by default.

`input_file`
: Specifies the name of the Server Description File.

Environment Variables

`TUXDIR`
: Finds the WLE libraries and include files to use when compiling the server application.

`LD_LIBRARY_PATH` (UNIX systems)
: Indicates which directories contain shared objects to be used by the compiler, in addition to the WLE shared objects. A colon (`:`) is used to separate the list of directories.

`LIB` (Windows NT systems)
: Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

Portability The `buildjavaserver` command is not supported on client-only WLE systems.

Example The following example builds a Java WLE server application `jar` file on a UNIX system. This example uses the `com/acme` path for locating classes and packages for the archive and also uses the Server Description File `MyServer.xml`.

```
buildjavaserver -s com/acme MyServer.xml
```

buildobjclient

Synopsis Constructs a WLE client application.

Syntax

```
buildobjclient [-v][-o name] [-f firstfile-syntax]
                [-l lastfile-syntax] -P
```

Description Use the `buildobjclient` command to construct a WLE client application. The command combines the files specified in the `-f` and `-l` options with the standard WLE libraries to form a client application. The client application is built using the default C++ language compile command defined for the operating system in use.

All specified `.c` and `.cpp` files are compiled in one invocation of the compilation system for the operating system in use. Users may specify the compiler to invoke by setting the `CC` environment variable to the name of the compiler. If the `CC` environment variable is not defined when `buildobjclient` is invoked, the default C++ language compile command for the operating system in use will be invoked to compile all `.c` and `.cpp` files.

Users may specify options to be passed to the compiler by setting the `CFLAGS` or the `CPPFLAGS` environment variables. If `CFLAGS` is not defined when `buildobjclient` is invoked, the `buildobjclient` command uses the value of `CPPFLAGS` if that variable is defined.

Options

`-v`

Specifies that the `buildobjclient` command should work in verbose mode. In particular, it writes the compile command to its standard output.

`-o name`

Specifies the name of the client application generated by this command. If the name is not supplied, the application file is named `client<.type>`, where type is an extension that is dependent on the operating system for an application (for example, on a UNIX system, there would not be a `type`; on a Windows NT system, the `type` would be `.EXE`).

`-f firstfile-syntax`

Specifies a file to be included first in the compile and link phases of the `buildobjclient` command. The specified file is included before the WLE libraries are included. There are three ways of specifying a file or files, as shown in Table 1-1.

Table 1-1 Specifying the First Filename(s)

Filename Specification	Definition
`-f firstfile`	One file is specified.
`-f "file1.cpp file2.cpp file3.cpp ..."`	Multiple files may be specified if they are enclosed in quotation marks and are separated by white space.

Note: Filenames that include spaces are not supported

Note: The `-f` option may be specified multiple times.

`-l lastfile-syntax`

Specifies a file to be included last in the compile and link phases of the `buildobjclient` command. The specified file is included after the WLE libraries are included. There are three ways of specifying a file, as shown in Table 1-2.

Table 1-2 Specifying the Last Filename(s)

Filename Specification	Definition
`-l lastfile`	One file is specified.
`-l "file1.cpp file2.cpp file3.cpp ..."`	Multiple files may be specified if they are enclosed in quotation marks and are separated by white space.

Note: The `-l` option may be specified multiple times.

`-P`

Specifies that the appropriate POA libraries should be linked into the image (that is, libraries that allow a client to also function as a server). The resulting image can act as a server and can use the `Callbacks` wrapper class for creating objects. The resulting joint client/server cannot take advantage of the object state management and transaction management provided by the WLE TP Framework. The `-P` switch should have been passed to the IDL compiler

when generating the client. Use `buildobjserver` to build a server with all the support provided by the TP Framework. The default is to not link in the server libraries; that is, the default is to create a client only, not a joint client/server.

`-h or -?`
Provides help that explains the usage of the `buildobjclient` command. No other action results.

Environment Variables

`TUXDIR`
Finds the WLE libraries and include files to use when compiling the client applications.

`CC`
Indicates the compiler to use to compile all files with `.c` or `.cpp` file extensions. If not defined, the default C++ language compile command for the operating system in use will be invoked to compile all `.c` and `.cpp` files.

`CFLAGS`
Indicates any arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions. If `CFLAGS` does not exist in the `buildobjclient` command environment, the `buildobjclient` command checks for the `CPPFLAGS` environment variable.

`CPPFLAGS`

Note: Arguments passed by the `CFLAGS` environment variable take priority over the `CPPFLAGS` variable.

Contains a set of arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions.
This is in addition to the command line option `"-I$(TUXDIR)/include"` for UNIX systems or the command line option `/I%TUXDIR%\include` for Windows NT systems, which is passed automatically by the `buildobjclient` command. If `CPPFLAGS` does not exist in the `buildobjclient` command environment, no compiler commands are added.

`LD_LIBRARY_PATH` (UNIX systems)
Indicates which directories contain shared objects to be used by the compiler, in addition to the objects shared by the WLE software. A colon (:) is used to separate the list of directories.

`LIB` (Windows NT systems)
Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

Portability The `buildobjclient` command is not supported on client-only platforms.

Examples The following example builds a WLE client application on an NT system:

```
set CPPFLAGS=-I%APPDIR%\include
buildobjclient -o empclient.exe -f emp_c.cpp -l userlib1.lib
```

The following example builds a WLE client application on a UNIX system using the c shell:

```
setenv CPPFLAGS=$APPDIR/include
buildobjclient -o empclient -f emp_c.cpp -l userlib1.a
```

buildobjserver

Synopsis Constructs a WLE server application.

Syntax

```
buildobjserver [-v] [-o name] [-f firstfile-syntax]
       [-l lastfile-syntax] [-r rmname]
```

Description Use the `buildobjserver` command to construct a WLE server application. The command combines the files specified in the `-f` and `-l` options with the main routine and the standard WLE libraries to form a server application. The server application is built using the default C++ compiler provided for the platform.

All specified `.c` and `.cpp` files are compiled in one invocation of the compilation system for the operating system in use. Users may specify the compiler to invoke by setting the `CC` environment variable to the name of the compiler. If the `CC` environment variable is not defined when `buildobjserver` is invoked, the default C++ language compile command for the operating system in use will be invoked to compile all `.c` and `.cpp` files.

Users may specify options to be passed to the compiler by setting the `CFLAGS` or the `CPPFLAGS` environment variables. If `CFLAGS` is not defined when `buildobjserver` is invoked, the `buildobjserver` command uses the value of `CPPFLAGS` if that variable is defined.

Options

`-v`

Specifies that the `buildobjserver` command should work in verbose mode. In particular, it writes the compile command to its standard output.

`-o name`

Specifies the name of the server application generated by this command. If the name is not supplied, the application file is named `server<.type>`, where type is the extension that is dependent on the operating system for an application (for example, on UNIX systems, there would not be a `type`; on Windows NT systems, the `type` would be `.EXE`).

`-f firstfile-syntax`

Specifies a file to be included first in the compile and link phases of the `buildobjserver` command. The specified file is included before the WLE libraries are included. For a description of the three ways to specify a file or files, see Table 1-1, “Specifying the First Filename(s),” on page 1-6.

`-l lastfile-syntax`

Specifies a file to be included last in the compile and link phases of the `buildobjserver` command. The specified file is included after the WLE

libraries are included. For a description of the three ways to specify a file or files, see Table 1-2, "Specifying the Last Filename(s)," on page 1-6.

`-r rmname`

Specifies the resource manager associated with this server. The value `rmname` must appear in the resource manager table located in `$TUXDIR/udataobj/RM` on UNIX systems or `%TUXDIR%\udataobj\RM` on Windows NT systems. Each entry in this file is of the form `rmname:rmstructure_name:library_names`. Using the `rmname` value, the entry in `$TUXDIR/udataobj/RM` or `%TUXDIR%\udataobj\RM` automatically includes the associated libraries for the resource manager and properly sets up the interface between the transaction manager and the resource manager. The value `TUXEDO/SQL` includes the libraries for the BEA Tuxedo System/SQL resource manager. Other values can be specified as they are added to the resource manager table. If the `-r` option is not specified, the default is to use the null resource manager.

`-h or -?`

Provides help that explains the usage of the `buildobjserver` command. No other action results.

Environment Variables

`TUXDIR`

Finds the WLE libraries and include files to use when compiling the server application.

`CC`

Indicates the compiler to use to compile all files with `.c` or `.cpp` file extensions that are passed in through the `–l` or `-f` options.

`CFLAGS`

Specifies any arguments that are passed as part of the compiler command line for any files with `.c` or `.cpp` file extensions. If `CFLAGS` does not exist in the `buildobjserver` command environment, the buildobjserver command checks for the `CPPFLAGS` environment variable.

`CPPFLAGS`

Note: Arguments passed by the `CFLAGS` environment variable take priority over the `CPPFLAGS` environment variable.

Contains a set of arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions.This is in addition to the command line option `-I$(TUXDIR)/include` for UNIX systems or the command line option `/I%TUXDIR%\include` for Windows NT systems,

which is passed automatically by the `buildobjserver` command. If `CPPFLAGS` does not exist in the `buildobjserver` command environment, no compiler commands are added.

`LD_LIBRARY_PATH` (UNIX systems)
: Indicates which directories contain shared objects to be used by the compiler, in addition to the WLE shared objects. A colon (:) is used to separate the list of directories.

`LIB` (Windows NT systems)
: Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

Portability

The `buildobjserver` command is not supported on client-only WLE systems.

Examples

The following example builds a WLE server application on a UNIX system using the `emp_s.cpp` and `emp_i.cpp files`:

```
buildobjserver -r TUXEDO/SQL -o unobserved
                -f “emp_s.cpp  emp_i.cpp”
```

The following example shows how to use the `CC` and `CFLAGS` environment variables with the `buildobjserver` command. The example also shows how to link in the math library on UNIX systems using the Bourne or Korn shells using the `-f` and `-lm` options:

```
CFLAGS=-g CC=/bin/cc \
buildobjserver -r TUXEDO/SQL -o TLR -f TLR.o -f util.o -l -lm
```

The following example shows how to use the `buildobjserver` command on UNIX systems with no resource manager specified:

```
buildobjserver -o PRINTER -f PRINTER.o
```

Sample RM Files

The following are sample RM files for all the supported operating system platforms:

Windows NT

```
Oracle_XA;xaosw;C:\Orant\rdbms73\xa\xa73.lib
    C:\Orant\pro22\lib\msvc\sqllib18.lib
```

UNIX

```
Oracle_XA:xaosw:-L$ORACLE_HOME/rdbms/lib
    -L$ORACLE_HOME/precomp/lib -lc
   -L/home4/m01/app/oracle/product/7.3.2/lib -lsql -lclntsh
    -lsqlnet -lncr -lcommon -lgeneric -lepc -lnlsrtl3 -lc3v6
```

```
    -lcore3 -lsocket -lnsl -lm -ldl -lthread
```

Digital UNIX

```
Oracle_XA:xaosw:-L${ORACLE_HOME}/lib -lxa
    ${ORACLE_HOME}/lib/libsql.a -lsqlnet -lncr -lsqlnet
    ${ORACLE_HOME}/lib/libclient.a -lcommon -lgeneric -lsqlnet
    -lncr -lsqlnet ${ORACLE_HOME}/lib/libclient.a -lcommon
    -lgeneric -lepc -lepcpt -lnlsrtl3 -lc3v6 -lcore3
    -lnlsrtl3 -lcore3 -lnlsrtl3 -lm
```

AIX

```
Oracle_XA:xaosw:-L${ORACLE_HOME}/lib -lxa -lsql -lsqlnet
    -lncr -lclient -lcommon -lgeneric -lepc -lnlsrtl3 -lc3v6
    -lcore3 -lm -lld
```

HP-UX : Oracle 8.04

```
    Oracle_XA:xaosw:-L${ORACLE_HOME}/lib -lclntsh
```

buildtms

See the description of the `buildtms` command in the *BEA Tuxedo Reference manual*.

buildXAJS

Synopsis Constructs an XA resource manager to be used with a Java server application group.

Syntax `buildXAJS [-v] -r rmname [-o outfile]`

Description Use this command to build an XA resource manager that you want to use with a Java server application group. In the application's `UBBCONFIG` file, you use the `JavaServerXA` element in place of the `JavaServer` element to associate the XA resource manager with a specified server group. Note that a server application configured to use the default XA resource manager (that is, `NULL`) cannot coexist in a server group that uses a nondefault XA resource manager, such as Oracle. Refer to the *Administration Guide* for more information about configuring server groups with an XA resource manager.

Options

`-v`

Specifies that the `buildXAJS` command should work in verbose mode. In particular, it writes the build command to its standard output.

`-r rmname`

Specifies the resource manager associated with this server. The value `rmname` must appear in the resource manager table located in `$TUXDIR/udataobj/RM` on UNIX systems, or `%TUXDIR%\udataobj\RM` on Windows NT systems. On UNIX systems, each entry in this file is of the form `rmname:rmstructure_name:library_names`. On NT systems, each entry in this file is of the form `rmname;rmstructure_name;library_names`. Using the `rmname` value, the entry in `$TUXDIR/udataobj/RM` or `%TUXDIR%\udataobj\RM` automatically includes the associated libraries for the resource manager and properly sets up the interface between the transaction manager and the resource manager. The value `TUXEDO/SQL` includes the libraries for the BEA Tuxedo System/SQL resource manager. Other values can be specified as they are added to the resource manager table. If the `-r` option is not specified, the default is to use the null resource manager.

`-o outfile`

Specifies the name of the output file. If no name is specified, the default is `JavaServerXA`.

Environment Variables `TUXDIR`

Finds the WLE libraries and include files to use when compiling the server application.

`LD_LIBRARY_PATH` (UNIX systems)
: Indicates which directories contain shared objects to be used by the compiler, in addition to the WLE shared objects. A colon (:) is used to separate the list of directories.

`LIB` (Windows NT systems)
: Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

Portability The `buildXAJS` command is not supported on client-only WLE systems.

Example The following example builds a Java server XA resource manager on a UNIX system:

```
buildXAJS -r oracle7
```

ejbc

Synopsis Produces a deployable EJB JAR file.

Syntax `java com.beasys.ejb.utils.ejbc` *`options jar-file archive-files`*

Description The `ejbc` command produces a deployable EJB JAR file. You can also use this command to generate a standard EJB JAR file for distribution. The primary input to the `ejbc` command is a standard EJB deployment descriptor file specified in XML, and optionally an XML deployment descriptor file specifying the WebLogic EJB extensions to the deployment descriptor DTD.

The `ejbc` command performs the following steps:

1. Parses the standard EJB deployment descriptor XML file and the WebLogic EJB extensions to the deployment descriptor DTD. You can provide this input as separate input files, or as an existing EJB JAR file. The XML file names are `ejb-jar.xml` and `weblogic-ejb-extensions.xml`. If you use the `–nodeploy` option, the file containing the WebLogic EJB extensions to the deployment descriptor DTD, if specified, is ignored.
2. Checks the deployment descriptors for semantic consistency, and writes any inconsistencies to standard output.
3. Generates the wrapper java classes and compiles them. This is performed for each EJB in the deployment descriptor. Note that the `ejbc` command does not compile the bean classes; the bean classes must be compiled before you use the `ejbc` command.

   If you specify the `–nodeploy` option, the `ejbc` command does not generate any wrapper classes.
4. Packages the XML deployment descriptors and the generated class files into a deployable EJB JAR file. The `ejbc` command ignores the archive arguments; the `ejbc` command simply packages the contents of the destination directory (which may be the current dirrectory) and the standard deployment descriptor and WebLogic EJB extensions to the deployment descriptor DTD files into the EJB JAR file.

Options `-help`

Prints a short description of the arguments for the command.

`-i` *`input-file`*

Specifies the input file, which can be either an existing EJB JAR file or a standard EJB deployment descriptor file. If you do not specify an input file, the `ejbc` command checks for an existing EJB JAR file and uses that for input. You must name the standard EJB deployment descriptor file `ejb-jar.xml`.

`-x` *`wldd-file`*

Optional. Identifies the WebLogic EJB extensions to the deployment descriptor DTD. If you do not specify this file, but you do provide an input EJB JAR file, the `ejbc` command attempts to read the WebLogic EJB extensions to the deployment descriptor DTD from the EJB JAR file instead. You must name the extended EJB deployment descriptor file `wlejb-jar.xml`.

`-classpath` *`path`*

Sets the `CLASSPATH` for the `ejbc` command. This overrides the system or shell `CLASSPATH`.

`-d` *`directory`*

Sets the destination directory for the generated class files. If you do not enter this option, the `ejbc` command uses the current directory.

`-compiler` *`javac`*

Sets the Java compiler. The default is `javac`.

`-keepgenerated`

Saves the intermediate Java files generated for deployment.

`-nodeploy`

Creates a standard EJB JAR file. When you use this option, the `ejbc` command does not generate wrapper classes, and only processes standard XML. If a file specifying the WebLogic EJB extensions to the deployment descriptor DTD is present in the input to the `ejbc` command, that XML descriptor is ignored and is not written to the archive.

`jar-file`

Optional. This specifies the output deployable EJB JAR file for the WebLogic Enterprise JavaServer. This file also serves as the default input for the deployment descriptors, if you do not specify them explicitly using the `-i` or `-x` option. If you do not specify this argument, any generated class files are placed only in the destination directory.

archive-files

Identifies the files to be included in the output EJB JAR file. This argument uses the same syntax as the standard JDK `jar` utility, and can specify one or more files or directories. If any of the files is a directory, the `ejbc` command processes the directory recursively. You may use wildcards in the file specification. This `archive-files` argument is optional. If you do not specify this argument, the `ejbc` command places the generated classes in the destination directory (specified with the `-d` option).

genicf

Synopsis Generates an ICF file.

Syntax `genicf [options] idl-filename...`

Description Given the `idl-filename(s)`, generates an ICF file that provides the code generation process with additional information about policies on implementations and the relationship between implementations and the interface they implement. If an ICF file is provided as input to the `idl` command, the `idl` command generates server code for only the implementation/interface pairs specified in the ICF file.

The generated ICF file has the same filename as the first `idl-filename` specified on the command line, but with a `.icf` extension.

If incorrect OMG IDL syntax is specified in the `idl-filename(s)` file, appropriate errors are returned.

Options

`-D identifier=[definition]`
Performs the same function as the `#define` C++ preprocessor directive; that is, the `-D` option defines a token string or a macro to be substituted for every occurrence of a given identifier in the definition file. If a definition is not specified, the identifier is defined as 1. Multiple `-D` options can be specified. White space between the `-D` option and the identifier is optional.

`-I pathname`
Specifies directories within which to search for include files, in addition to any directories specified with the `#include` OMG IDL preprocessor directive. Multiple directories can be specified by using multiple `-I` options.

There are two types of `#include` OMG IDL preprocessor directives: system (for example, `<a.idl>`) and user (for example, `"a.idl"`). On UNIX systems, the path for system `#include` directories is `/usr/include` and any directories specified with the `-I` option; the path for user `#include` directives is the location of the file containing the `#include` directive, followed by the path specified for the system `#include` directive. On Windows NT systems, no distinction is made between the system `#include` directories and the user `#include` directives.

`-h and -?`
Provides help that explains the usage of the `genicf` command. No other action results.

Example This command creates the `emp.icf` file: `genicf emp.idl`.

idl

Synopsis

Compiles the Object Management Group (OMG) Interface Definition Language (IDL) file and generates the files required for the interface.

Syntax

```
idl [-i] [-Did[=value]] [-I pathname][-h] [-P] [-T] idl-filename...
          [ icf-filename...]
```

Description

Given the provided `idl-filename()` file(s) and optional `icf-filename()` file(s), the `idl` command generates the following files:

`idl-filename_c.cpp`
: Client stub (includes embedded user-defined data type functions).

`idl-filename_c.h`
: Class definitions for interfaces.

`idl-filename_s.cpp`
: Server skeleton containing an implementation of the `POA_skeleton` classes.

`idl-filename_s.h`
: `POA_skeleton` class definitions.

`idl-filename_i.cpp`
: Example version of the implementation. This file is generated only when the `-i` option is given.

`idl-filename_i.h`
: Class definition of an example implementation that inherits from the `POA_skeleton` class. This file is generated only when the `-i` option is given.

Note: If any ICF files are specified, the information within the ICF files is used to provide the code generator with information about the interface/implementations that override the defaults. Typically, an activation policy and a transaction policy for an implementation may be specified in the ICF file. If no ICF files are specified, default policies are in effect for all of the interfaces specified in the OMG IDL file, and skeleton code for all of the interfaces is generated. If an `icf-filename` is provided as input to the `idl` command, only the implementation/interfaces specified in the `icf-filename` are generated as part of the server.

The IDL compiler places the generated client stub information in the `filename_c.cpp` and `filename_c.h` files. The generated server skeleton information is placed in the `filename_s.cpp` and `filename_s.h` files.

The IDL compiler overwrites the generated client stub files (`filename_c.cpp` and `filename_c.h`), and the generated server skeleton files (`filename_s.cpp` and `filename_s.h`). Any previous versions are destroyed.

When using the `-i` option, the IDL compiler overwrites the sample implementation class definition file (`filename_i.h`). Previous versions are destroyed. The sample implementation file (`filename_i.cpp`) is overwritten, however, any code contained within the code preservation blocks is preserved and restored in the newly generated file. To avoid the loss of data, it is recommended that you copy the sample implementation files (`filename_i.h` and `filename_i.cpp`) to a safe location before regenerating these files.

If an unknown option is passed to this command, the offending option and a usage message is displayed to the user, and the compile is not performed.

Parameter

`idl filename`

The name of one or more files that contain OMG IDL statements.

Options

`-D identifier[=definition]`

Performs the same function as the `#define C++` preprocessor directive; that is, the `-D` option defines a token string or a macro to be substituted for every occurrence of a given identifier in the definition file. If a definition is not specified, the identifier is defined as 1. Multiple `-D` options can be specified. White space between the `-D` option and the name is optional.

`-I pathname`

Specifies directories within which to search for include files, in addition to any directories specified with the `#include` OMG IDL preprocessor directive. Multiple directories can be specified by using multiple `-I` options.

There are two types of `#include` OMG IDL preprocessor directives: `system` (for example, `<a.idl>`) and `user` (for example, `"a.idl"`). The path for system `#include` directories is the system include directory and any directories specified with the `-I` option. The path for user `#include` directives is the location of the file containing the `#include` directive, followed by the path specified for the system `#include` directive.

By default, the text in files included with an `#include` directive is not included in the client and server code that is generated.

`-i`

Results in `idl-filename_i.cpp` files being generated. These files contain example templates for the implementations that implement the interfaces specified in the OMG IDL file.

Note: When using the `idl` command `-i` option to update your implementation files, proceed as follows to update your implementation files:

1. Back up your implementation files.
2. If you are migrating from BEA ObjectBroker to WLE, edit your generated implementation files to change the code preservation block delimiters from `"OBB_PRESERVE_BEGIN"` and `"OBB_PRESERVE_END"` to `"M3_PRESERVE_BEGIN"` and `"M3_PRESERVE_END"`.
3. If you added include files to your method implementation file (`*_i.cpp`), edit the file and move the includes inside the `INCLUDES` preservation block.
4. Regenerate your edited implementation files (using the `idl` command with the `-i` option).
5. If you previously made modifications to the implementation definition file (`*_i.h`), edit the newly generated definition file and add your modifications back in. Be sure to put your modifications inside the code preservation blocks so subsequent updates will automatically retain them. Pay particular attention to the implementation constructor and destructor functions; the function signatures have been changed in this release.
6. If you previously made modifications outside the preservation blocks of the method implementation file (`*_i.cpp`) or to the implementation constructor and destructor functions, edit the newly generated file and add those modifications. Be sure to put the modifications inside a preservation block so subsequent updates will automatically retain them.

`-P`

Generates server code that uses the POA instead of the TP Framework. With this option specified, the skeleton class does not inherit from the TP Framework `Tobj_ServantBase` class, but instead inherits directly from the `PortableServer::ServantBase` class. By default, the skeleton class uses the TP Framework. So you must use this switch when you are developing joint client/servers as these servers do not use the TP framework.

Not having the `Tobj_ServantBase` class in the inheritance tree for a servant means that the servant does not have `activate_object` and

`deactivate_object` methods. In WLE servers these methods are called by the TP Framework to dynamically initialize and save a servant's state before invoking a method on the servant. For WLE joint client/servers, user-written code must explicitly create a servant and initialize a servant's state; therefore, the `Tobj_ServantBase` operations are not needed. When using the `-P` option, ICF files are not used because the TP Framework is not available.

`-T`

Generates tie-based servant code that allows the use of delegation to tie an instance of a C++ implementation class to the servant. This option allows classes that are not related to skeletons by inheritance to implement CORBA object operations. This option is set to off by default.

-h or -?

Provides help that explains the usage of the `idl` command. No other action results.

Examples

```
idl emp.idl
idl emp.idl emp.icf
```

idl2ir

Synopsis Creates the Interface Repository and loads interface definitions into it.

Syntax

```
idl2ir [options] definition-filename-list
```

Options The options are as follows:

```
[-f repository-name] [-c]
[-D identifier[=definition]]
[-I pathname [-I pathname] [...]] [-N{i|e}]
```

Description Use this command to create the Interface Repository and to load it with interface definitions. If no repository file exists, this command creates it. If a repository file does exist, this command loads the specified interface definitions into it and, in effect, updates the file.

One of the side effects of doing this is that a new Interface Repository database file is created.

Parameters

`definition-filename-list`
A list of file specifications containing the repository definitions. These files are treated as one logical file and are loaded in one operation.

`-f repository-name`
The filename of the Interface Repository file. If you do not specify the `-f` option, the `idl2ir` command creates `repository.ifr as the` Interface Repository file on UNIX systems and `repository_1.ifr` on Microsoft Windows NT systems.

-c
Indicates that a new repository is to be created. If a repository exists and this option is specified, the `idl2ir` command ignores the existing repository and replaces it with a new one. If a repository exists and this option is not specified, the `idl2ir` command updates the existing repository.

-D identifier[=definition]
Performs the same function as the `#define` preprocessor directive; that is, the `-D` option defines a token string or a macro to be substituted for every occurrence of a given identifier in the definition file. If a definition is not specified, the identifier is defined as 1.You can specify multiple `-D` options.

`-I pathname`

Specifies a directory within which to search for include files, in addition to any directories specified with the `#include` OMG IDL preprocessor directive.

There are two types of `#include` OMG IDL preprocessor directives: system (for example, `<a.idl>`) and user (for example, `"a.idl"`). The path for system `#include` directives is `/usr/include` for UNIX systems, and any directories specified with the `-I` option. The path for system `#include` directives is the local directory for Windows NT systems, and any directories specified with the -I option.

The path for user `#include` directives is the current directory and any directories specified with the `-I` option. Multiple `-I` options can be specified.

Note: Additional definitions loaded into the interface repository while the server process for the Interface Repository is running are not accepted until the server process for the Inteface Repository is stopped and started again.

idltojava

Synopsis

Compiles IDL files to Java source code based on IDL to Java mappings defined by the OMG.

The `idltojava` compiler provided with BEA WebLogic Enterprise (WLE) includes several enhancements, extensions and additions that are not present in the original Sun Microsystems, Inc. version of the compiler. The WebLogic Enterprise specific revisions are summarized here.

The BEA WebLogic Enterprise `idltojava` compiler:

- Differs from that described in the Sun Microsystems, Inc. documentation in behavior and defaults of the flags.
- Includes a new #pragma tag: `#pragma ID <name> <Repostitory_id>`
- Includes a new #pragma tag: `#pragma version <name> <m.n>`
- Extends the `#pragma prefix` to work on inner scope. A blank prefix reverts.
- Allows unions with boolean discriminators
- Allow declarations nested inside complex types

Syntax

```
idltojava [idltojava Command Flags] [idltojava Command Options]
filename ...
```

```
m3idltojava [idltojava Command Flags] [idltojava Command Options]
filename ...
```

To run `idltojava` on Client-side IDL files, use the following command:

`idltojava <flags> <options> <idl-files>`

The `idltojava` command requires a C++ pre-processor, and is used to generate deprectated names. The command `idltojava` generates Java code as is appropriate for the client-side ORB.

Note: A remote *joint client/server* is a client that implements server objects to be used as callback objects. The server role of the remote joint client/server is considerably less robust than that of a WebLogic Enterprise server. Neither the client nor the server has any of the WebLogic Enterprise administrative and infrastructure components, such as `tmadmin`, JNDI registration, and ISL/ISH (hence, none of scalability and reliability attributes of WebLogic Enterprise)

To run `m3idltojava` on Server-side IDL files, use the following command:

`m3idltojava <flags> <options> <idl-files>`

The server-side ORB is built to use non-deprecated names. The command `m3idltojava` generates Java code using non-deprecated names as is appropriate for the server-side ORB.

Description

The `idltojava` command compiles IDL source code into Java source code. You then use the `javac` compiler to compile that source to Java bytecodes.

The command `idltojava` is used to translate IDL source code into generic client stubs and generic server skeletons which can be used for callbacks. The command `m3idltojava` is used to translate IDL into generic client stubs and WebLogic Enterprise server skeletons.

The IDL declarations from the named IDL files are translated to Java declarations according to the mappings specified in the OMG IDL to Java mappings.

Options

Note: Several option descriptions have been added here that are not documented in the original Sun Microsystems Inc. `idltojava` compiler documentation.

Option	Description
-j `javaDirectory`	Specifies that generated Java files should be written to the given *directory*. This directory is independent of the -p option, if any.
-J `filesFile`	Specifies that a list of the files generated by `idltojava` should be written to *filesFile*
-p `package-name`	Specifies the name of an outer package to enclose all the generated Java. It has the same function as `#pragma javaPackage`. **Note:** You must include an *outer package*. The compiler does not do this for you. If you do not have an outer package, the `idltojava` compiler will still generate Java files for you but you will get a Java compiler error when you try to compile the `*.java` files.
The following options are identical to the equivalent C/C++ compiler options (cpp):	
-I`directory`	Specifies a directory or path to be searched for files that are *#include*d in IDL files. This option is passed to the preprocessor.

Option	Description
-Dsymbol	Specifies a symbol to be defined during preprocessing of the IDL files. This option is passed to the preprocessor.
-Usymbol	Specifies a symbol to be undefined during preprocessing of the IDL files. This option is passed to the preprocessor.

Command Flags

The flags can be turned on by specifying them as shown, and they can be turned off by prefixing them with the letters **no-**. For example, to prevent the C preprocessor from being run on the input IDL files, use **-fno-cpp**.

The table below includes descriptions of all flags.

Flag	Description
-flist-flags	Requests that the state of all the **-f** flags be printed. The default value of this flag is **off**.
-flist -debug-flags	Provides a list of debugger flags
-fcaseless	Request that case not be significant in keywords and identifiers. The default value of this flag is 'on'.
-fclient	Requests the generation of the client side of the IDL files supplied. The default value of this flag is 'off'.
-fcpp	Requests that the idl source be run through the C/C++ preprocessor before being compiled by the idltojava compiler. The default value of this flag is **on**.
-fignore-duplicates	specifies that duplicate definitions be ignored. This may be useful if compiling multiple idl files at one time. The default value of this flag is **off**.
-flist-options	Lists the options specified on the command line. The default value of this flag is **off**.
-fmap-included-files	Specifies that java files be generated for definitions included by #include preprocessor directives. The default value for this flag is **off** which specifies that the java files for included definitions **not** be generated.
-fserver	Requests the generation of the server side of the IDL files supplied. The default value of this flag is **off**.

-fverbose	Requests that the compiler comment on the progress of the compilation. The default value of this flag is **off**.
-fversion	Requests that the compiler print its version and timestamp. The default value of this flag is **off**.
-fwarn-pragma	Requests that warning messages be issued for unknown or improperly specified #pragma's. The default value of this flag is **on**.
-fwrite-files	Requests that the derived java files be written. The default value of this flag is 'on'. You might specify -fno-write-files if you wished to check for errors without actually writing the files.

Notes

The BEA WebLogic Enterprise idltojava compiler processes #pragma somewhat differently from the Sun Microsystems, Inc. idltojava compiler.

RepositoryPrefix="prefix"

A default repository prefix can also be requested with the line #pragma prefix "requested prefix" at the top-level in the IDL file itself.

#pragma javaPackage "package"

Wraps the default package in one called package. For example, compiling an IDL module M normally creates a Java package M. If the module declaration is preceded by:

#pragma javaPackage browser

the compiler will create the package M inside package browser. This pragma is useful when the definitions in one IDL module will be used in multiple products. The command line option -p can be used to achieve the same result

#pragma ID scoped-name "IDL:<path>:<version>"

specifies the repository ID of the identifier scoped-name. This pragma may appear any where in an IDL file. If the pragma appears inside a complex type such as structure or union then only as much of scoped-name need be specified to specify the element. A scoped-name is of the form outer_name::name::inner_name. The <path>component of the repository id is a series of identifiers separated by forward slashes (/). The <version>component is a decimal number MM.mm, where MM is the major version number and mm is the minor version number.

ir2idl

Synopsis Shows the contents of an Interface Repository.

Syntax

```
ir2idl [options] [interface-name]
```

Options The options are as follows:

```
[-f repository-name] [-n]
[-t interface-type] [-o filename]
```

Description This command shows the contents of an Interface Repository. By directing the output to a file with the `-o` option, you can extract the OMG IDL file from the repository. By default, the repository file is `repository.ifr`.

Parameters

`interface-name`
: The name of the interface whose contents are to be shown. If you do not specify an interface name, all interfaces in the repository are shown.

`-f repository-name`
: The name of the repository to search for the interface definitions. If you do not specify the `-f` option, `repository.ifr` is used.

`-n`
: Specifies that the output should not include those objects that were inherited.

`-t interface-type`
: Indicates the type of objects to display. The object type must be one of the following keywords:

```
Attribute
Constant
Exception
Interface
Method
Module
Operation
Typedef
```

If you do not specify this option, the default is to display all of the types.

`-o filename`
: The file specification for the file in which to write the retrieved OMG IDL statements. The default is standard output.

irdel

Synopsis Deletes the specified object from an Interface Repository.

Syntax

```
irdel [-f repository-name]  [-i id] object-name
```

Description This command deletes the specified interface from the repository. Only interfaces not referenced from another interface can be deleted. By default, the repository file is `repository.ifr`.

Parameters

`-f repository-name`
: An optional parameter that specifies an Interface Repository. The `repository-name` value is the file specification of an Interface Repository. If this option is not specified, the `repository.ifr` is used as the default.

`-i id`
: The repository `id` for the specified object. The `id` is used as a secondary level of lookup. If the `id` does not match the `id` of the named object, the object is not deleted.

`object-name`
: The name of the interface to delete from the repository. The name can be a simple object name or a scoped name, for example, `MOD1::INTERF2::OP3` (operation `OP3` is within interface `INTERF2`, which is in application `MOD1`).

ISL

Synopsis Enables access to WLE objects by remote WLE clients using IIOP.

Syntax

```
ISL SRVGRP="identifier"

    SRVID="number"

    CLOPT="[ -A ] [ servopts options ] -- -n netaddr
           [-a]
           [ -C {detect|warn|none} ]
           [ -d device ]

           [-E principal_name]
           [ -K {client|handler|both|none} ]
           [ -m minh ]
           [ -M maxh ]
           [ -T Client-timeout]
           [ -x mpx-factor ]
           [-H external-netaddr]
           [-O]
           [-o outbound-max-connections]
           [-s Server-timeout]
           [-u out-mpx-users]”
           [-R renegotiation-interval]
           [-S secure port]
           [-v {detect|warn|none} ]
           [-z [0|40|56|128]]
           [-z [0|40|56|128]]
```

Description The IIOP Server Listener (ISL) is a WLE-supplied server command that enables access to WLE objects by remote WLE clients using IIOP. The application administrator enables access to the application objects by specifying the IIOP Server Listener as an application server in the `SERVERS` section. The associated command-line options are used to specify the parameters of the IIOP Server Listener and IIOP Server Handlers.

The location, server group, server `ID`, and other generic server-related parameters are associated with the ISL using the standard configuration file mechanisms for servers. ISL command-line options allow for customization.

Each ISL booted as part of an application facilitates application access for a large number of remote WLE clients by providing access via a single, well-known network address. The IIOP Server Handlers are started and stopped dynamically by the ISL, as necessary, to meet the incoming load.

For joint client/servers, if the remote joint client/server ORB supports bidirectional IIOP connections, the ISL can use the same inbound connection for outbound invokes to the remote joint client/server. The ISL also allows outbound invokes (outbound IIOP) to objects located in a joint client/server that is not connected to an ISH. This capability is enabled when the `-O` option is specified. The associated command-line options (those shown above in **boldface text**) allow configuration of outbound IIOP support:

Parameters

`-A`

Indicates that the ISL is to be booted to offer all its services. This is a default, but it is shown to emphasize the distinction between system-supplied servers and application servers. The latter can be booted to offer only a subset of their available services. The double-dash (`--`) marks the beginning of parameters that are passed to the ISL after it has been booted.

You specify the following options in the `CLOPT` string after the double-dash (`--`) in the `CLOPT` parameters:

`-n netaddr`

Specifies the network address to be used by a server listener to accept connections from remote CORBA clients. The remote client must set the environment variable (`TOBJADDR`) to this value, or specify the value in the Bootstrap object constructor. See the *C++ Programming Reference* for details. This is the only required parameter.

TCP/IP addresses must be specified in one of the following two formats:

```
"//hostname:port_number"
"//#.#.#.#:port_number"
```

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

Note: The hostname must begin with a letter character.

In the second format, the `"#.#.#.#"` is the dotted decimal format. In dotted decimal format, each # must 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 listens for incoming requests. `port_number` can be a number between 0 and 65535 or a name. If `port_number` is a name, it must be found in the network services database on your local machine.

Note: The Java `Tobj_Bootstrap` object uses a `short` type to store the `port_number`. Therefore, you must use a `port_number` in the range of 0 to 32767 if you plan to support connections from Java clients.

Note: The network address that is specified by programmers in the Bootstrap constructor or in `TOBJADDR` must exactly match the network address in the application's `UBBCONFIG` file. The format of the address as well as the capitalization must match. If the addresses do not match, the call to the Bootstrap constructor will fail with a seemingly unrelated error message:

```
ERROR: Unofficial connection from client at
<tcp/ip address>/<port-number>:
```

For example, if the network address is specified as `//TRIXIE:3500` in the `ISL` command line option string, specifying either `//192.12.4.6:3500` or `//trixie:3500` in the Bootstrap constructor or in `TOBJADDR` will cause the connection attempt to fail.

On UNIX systems, use the `uname -n` command on the host system to determine the capitalization used. On Windows NT systems, see the host system's Network control panel to determine the capitalization used.

Note: Unlike the BEA Tuxedo system Workstation Listener (WSL), the format of the network addresses is limited to `//host:port`. The reason for this limitation is that the host name and port number are used by WLE servers; the host name does not appear as such in the hexadecimal format, and it could only be passed to the servers using the dotted IP address format.

`[-a]`

Specifies that certificate-based authentication should be enabled when accepting an SSL connection from a remote application.

`[-C detect|warn|none]`

Determines how the IIOP Listener/Handler will behave when unofficial connections are made to it. The default value is detect.

The official way for the CORBA client to connect to the IIOP Listener/Handler is via a Bootstrap object. The unofficial connection is established directly from an IOR. For example, a client could connect to one IIOP Listener/Handler via a Bootstrap object and then, perhaps inadvertently, connect to a second IIOP Listener/Handler by using an IOR that contains the host and port of the second IIOP Listener/Handler. Typically, this is not the case. Usually, the client uses IORs that contain the host and port of the IIOP

Listener/Handler that the client connected to via a Bootstrap object. Use of such IORs does not cause an additional connection to be made.

Caution: The use of unofficial connections can cause problems for remote client applications that use transactions. The application may have the notion that invocations on both the official and unofficial connections within the same transaction have succeeded; however, in reality, only invocations on the official connection are ACID (Atomicity, Consistency, Isolation, and Durability).

A value of `detect` causes the ISL/ISH to raise a `NO_PERMISSION` exception when an unofficial connection is detected. A value of `warn` causes the ISL/ISH to log a message to the user log exception when an unofficial connection is detected; no exception will be raised. A value of `none` causes the ISL/ISH to ignore unofficial connections.

`[-d device]`

Specifies the device filename used for network access by the server listener and its server handlers. This parameter is optional because some transport providers (for example, sockets) do not require a device name. However, other providers (for example, TLI) do require a device name. In the case of TLI, this option is mandatory. There is no default for this parameter. (This does not apply to Windows NT systems.)

`[-E` *`principal_ name`*`]`

An optional parameter that indicates the identity of the principal that is required in order to establish a trusted connection pool. A trusted connection pool can only be established if a WebLogic Enterprise application is configured to require users to be authenticated.

If a remote client application attempts to propagate per-request security information over a connection that is not part of a trusted connection pool, the accompanying propagated security information will be ignored.

`[-K {client|handler|both|none}]`

Directs the client, or the ISH process, or both, to activate the network provider's `KEEPALIVE` option. This option improves the speed and reliability of network failure detection by actively testing an idle connection's state at the protocol stack level. The availability and timeout thresholds for this feature are determined by operating system tunable parameters.

A value of `client` configures this option for the client; a value of `handler` configures this option for the ISL; and a value of `both` will configure both

sides of the connection. The default value is none, in which case neither side has the KEEPALIVE option configured.

Note: The KEEPALIVE interval is an operating system parameter, so changing the value affects any other applications that enable KEEPALIVE. Many platforms have a two-hour default value that may be longer than desired.

This option is not available on all platforms. A userlog warning message is generated if the KEEPALIVE option is specified but is not available on the ISH's machine. If KEEPALIVE is requested but is not available on the client's machine, the setting is ignored.

`[-m minh]`

Specifies the minimum number of handlers that should be available in conjunction with this ISL at any given time. The default is 0. The The ISL will start this many ISHs immediately upon being booted and will not deplete the supply of ISHs below this number until the administrator issues a shutdown to the ISL. The default value for this parameter is 0. The legal range is between 0 and 255.

`[-M maxh]`

Specifies the maximum number of handlers that should be available in conjunction with this ISL at any given time. The Handlers are started as necessary to meet the demand of remote WLE clients attempting to access the system. The default value for this parameter is equal to the setting for MAXWSCLIENTS on the logical machine, divided by the multiplexing factor for this ISL (see -x option below), rounded up by one. The legal range for this parameter is between 1 and 4096. The value must be equal to or greater than minh.

`[-T` *`Client-timeout`*`]`

Specifies the inactive client timeout option. The inactive client timeout is the time (in minutes) allowed for a client to stay idle. If a client does not make any requests within this time period, the IIOP Listener/Handler disconnects the client. If this argument is not given or is set to 0, the timeout is infinite.

`[-x` *`mpx-factor`*`]`

This is an optional parameter used to control the degree of multiplexing desired within each ISH. The value for this parameter indicates the number of remote WLE clients that can be supported simultaneously by each ISH. The ISH ensures that new handlers are started as necessary to handle new remote WLE clients. This value must be greater than or equal to 1 and less than or equal to 4096. The default value for this parameter is 10.

`[-H external netadder]`

Specifies the external network address to be set as the host and port in interoperable object references returned to clients of the ISL. It has the same format as the ISL CLOPT `-n netaddr` option. This feature is useful when an IIOP, or remote, client needs to connect to an ISL through a firewall.

`[-O]`

This option (uppercase letter O) enables outbound IIOP to objects that are not located in a client that is connected to an ISH. Since the `–O` option requires a small amount of extra resources, the default is to not allow outbound IIOP.

`[-o outbound-max-connections]`

This option (lowercase letter o) specifies the maximum number of outbound connections that each ISH may have. In effect, it limits the number of simultaneous Outbound IIOP sockets that any single ISH under the control of this ISL will have active at one time.

This option requires that the `–O` (uppercase letter O) option is also specified. The value of this option must be greater than 0, but not more than 4096. An additional requirement is that the value of this option, (`outbound-max-connections`) times the maximum number of handlers, must be less than 32767. The default for this option is 20.

`[-R renegotiation-interval]`

Specifes the renegotiation internal in minutes. If a connection does not have a renegotiation in the specified interval, the IIOP Listener/Handler will request that the client renegotiate the session for inbound connections or actually perform the renegotiation in the case of outbound connections. The default is 0 minutes which results in no periodic session renegotiations.

`[-S secure-port]`

Specifies the port number that the IIOP Listener/Handler should use to listen for secure connections using the SSL protocol. You can configure the IIOP Listener/Handler to allow only secure connections by setting the port numbers specified by the `-S` and `-n` options to the same value.

`[-s Server-timeout]`

Server-timeout is the time, in minutes, allowed for a remote server to remain idle. If a remote server does not receive any requests within this time period, the ISL disconnects the outbound IIOP connection to the server. The ISH reconnects to the remote server on subsequent requests. This option can be used for server platforms that are unstable. Note that this is a best-attempt value in that the ISL does not disconnect the connection before this time is up,

but does not guarantee to disconnect the connection once the exact time has elapsed. This option requires that the `-O` (uppercase letter O) option is also specified. The value must be greater than or equal to 1. If this option is not specified, the default is 60 (one hour).

`[-u out-mpx-users]`

An optional parameter used to control the degree of outbound multiplexing desired within each ISH. The value for this option indicates the number of outbound IIOP users (native clients or servers) that can be supported simultaneously by each outbound IIOP connection in the ISH. The ISL ensures that new ISHs are started, as necessary, to handle new users up to the value of this option (`out-mpx-users`). This option requires that the –O (uppercase letter O) option is also specified. This option must be greater than 0 (zero), but not more than 1024; the default value is 10.

`[-v {detect|warn|none}]`

Determines how the IIOP Listener/Handler will behave when a digital certificate for a peer of an outbound connection initiated by the BEA object request broker (ORB) us received as part of the Secure Sockets Layer (SSL) protocol handshake. The validation is only performed by the initiator of a secure connection and confirms that the peer server is actually located at the same network address as specified by the domain name in the server's digital certificate. This validation is not technically part of the SSL protocol but is similar to the check done in web browsers.

A value of `detect` causes the BEA ORB to verify that the host specified in the object reference used to make the connection matches the domain name specified in the peer server's digital certificate. If the comparison fails, the BEA ORB refuses the authenticate the peer and drops the connection. The `detect` value is the default value.

A value of `warn` causes the BEA ORB to verify that the host specified in the object reference used to make the connection matches the domain name specified in the peer's digital certificate. If the comparison fails, the BEA ORB logs a message to the user log but continues to process the connection.

A value of none causes the BEA ORB to not perform the peer validation and to continue to process the connection.

The `-v` parameter is only available if the WebLogic Enterprise Security pack is installed.

`[-z |0|40|56|128]]`

Specifies the minimum level of encryption when establishing a network connection between a client and the IIOP Listener/Handler. 0 means no encryption while 40, 56, and 128 specify the length (in bits) of the encryption key. If this minimum level of encryption cannot be met, a connection will not be established. This option is only available if the WebLogic Enterprise Security Pack is installed.

`[-Z |0|40|56|128]]`

Specifies the maximum level of encryption when establishing a network connection between a client and the IIOP Listener/Handler. 0 means no encryption while 40, 56, and 128 specify the length (in bits) of the encryption key. The default is whatever capability is specified by the license. This option is only available if the WebLogic Enterprise Security Pack is installed

Portability The IIOP Server Listener is supported as a WLE-supplied server on UNIX and Microsoft Windows NT operating systems.

Interoperability The ISL works with any IIOP compliant ORB.

Depending on the type of remote object and the desired outbound IIOP configuration, you may have to perform additional programming tasks. lists the requirements for each type of object and outbound IIOP configuration.

Table 1-3 Programming Requirements for Using Outbound IIOP

Types of Objects	Asymmetric Requirements	Paired-connection Requirements	Bidirectional Requirements
Remote joint client/servers	Set `ISL CLOPT -O` option.	Use the `Tobj_Bootstrap::register_callback_port` method to register the callback port.	Use the `CORBA::ORB::create_policy` method to set `BiDirPolicy` on the POA.
Foreign (non-WLE) ORBs	Set `ISL CLOPT -O` option.	Not applicable.	If the foreign ORB supports the POA and `BiDirPolicy`, use the `CORBA::ORB::create_policy` method to set `BiDirPolicy` on the POA.
Remote clients	Remote clients are not servers, so outbound IIOP is not possible.		

Table 1-3 Programming Requirements for Using Outbound IIOP (Continued)

Native joint client/servers	Outbound IIOP is not used.
Native clients	Outbound IIOP is not used.

Network Addresses

Suppose the local machine on which the ISL is being run is using TCP/IP addressing and is named `backus.company.com`, with address `155.2.193.18`. Further suppose that the port number at which the ISL should accept requests is 2334. The address specified by the `-l` option could be:

```
//155.2.193.18:2334

//backus.company.com:2334
```

Examples

```
*SERVERS

ISL SRVGRP="ISLGRP" SRVID=1002 RESTART=Y GRACE=0

        CLOPT="-A -- -n //piglet:1900 -d /dev/tcp"
```

m3idltojava

Synopsis Compiles the Object Management Group (OMG) Interface Definition Language (IDL) file and generates client stub and server skeleton files required for the interface definitions being implemented in Java. Use this command only when you are creating a Java server application.

Syntax

```
m3idltojava [-p] [-j javaDirectory] [-Idirectory][-Dsymbol]
            [-Usymbol] [-foptions] idl-filename...
```

Description The `m3idltojava` command compiles OMG IDL source files into Java source code. You then use the javac compiler to compile that source into Java bytecodes. The OMG IDL declarations from the named OMG IDL files are translated to Java declarations according to the mapping from OMG IDL to Java.

Given the provided *`idl-filename`* file(s), the `m3idltojava` command generates the following files for each interface defined in the server application's OMG IDL file:

`interface-name``.java`
: Contains the Java version of the interface definitions in the OMG IDL file. Each interface implementation extends the `org.omg.CORBA.Object` class.

`_`*`interface-name`*`Stub.java`
: Is the client stub file.

`_`*`interface-name`*`ImplBase.java`
: Is the Server skeleton file, which is extended by the server application's object implementation classes.

`interface-name``Helper.java`
: Contains the helper class for the object.

`interface-name``Holder.java`
: Contains the holder class for the object.

The `m3idltojava` compiler generates the client stub and server skeleton files. Any previous versions are overwritten.

If an unknown option is passed to this command, the offending option and a usage message is displayed to the user, and the compile is not performed.

Parameter *`idl-filename`*
: Represents the name of one or more files that contain OMG IDL statements.

Options

`-p package`
: Specifies that generated Java classes should be part of the given package. The compiler creates the appropriate directory hierarchy and stores the generated files in the directory that corresponds to their package. If you specify the `-j` option, the hierarchy is created under the specified directory. Otherwise, the hierarchy is created under the current directory. You can override this option by using `#pragma javaPackage` in the OMG IDL source file.

`-j javaDirectory`
: Specifies that generated Java files should be written to the specified directory. This directory is independent of the `-p` option, if used.

`-Idirectory`
: Specifies directories within which to search for include files, in addition to any directories specified with the `#include` OMG IDL preprocessor directive. Multiple directories can be specified by using multiple `-I` options.

  There are two types of `#include` OMG IDL preprocessor directives: `system` (for example, `<a.idl>`) and `user` (for example, `"a.idl"`). The path for system `#include` directories is the system include directory and any directories specified with the `-I` option. The path for user `#include` directives is the location of the file containing the `#include` directive, followed by the path specified for the system `#include` directive.

  By default, the text in files included with an `#include` directive is not included in the client and server code that is generated.

`-Dsymbol`
: Specifies a symbol to be defined during OMG IDL file preprocessing. The `m3idltojava` command passes this symbol to the preprocessor.

`-Usymbol`
: Specifies a symbol to be undefined during OMG IDL file preprocessing. The `m3idltojava` command passes this symbol to the preprocessor.

`-foptions`
: You can enable the following options by specifying them as shown, and disable them by appending the string `no-`. For example, to prevent the C preprocessor from being run on the input OMG IDL files, specify `-fno-cpp`.

  `-flist-flags`
  : Displays the state of all -f flags. By default, this option is disabled.

  `-fclient`
  : Generates the client application files. By default, this option is enabled.

`-fserver`
: Generates the server application files. By default, this option is enabled.

`-fverbose`
: Specifies that the `m3idltojava` command should work in verbose mode. In particular, it writes command output to its standard output. By default, this option is disabled.

`-fversion`
: Specifies that the compiler prints its version and timestamp. By default, this option is disabled.

Examples

The following command generates only the server application files for `Simple.idl`:

```
m3idltojava -fno-client Simple.idl
```

The following command generates only the client application files for `Simple.idl`:

```
m3idltojava -fno-server Simple.idl
```

tmadmin

See the description of the `tmadmin` command in the *BEA Tuxedo Reference manual*.

tmboot

See the description of the `tmboot` command in the *BEA Tuxedo Reference manual*.

tmconfig

See the description of the `tmconfig` command in the *BEA Tuxedo Reference manual*.

tmloadcf

See the description of the `tmloadcf` command in the *BEA Tuxedo Reference manual*.

tmshutdown

See the description of the `tmshutdown` command in the *BEA Tuxedo Reference manual*.

tmunloadcf

See the description of the `tmunloadcf` command in the *BEA Tuxedo Reference manual*.

tpgrpadd

See the description of the `tpgrpadd` command in the *BEA Tuxedo Reference manual*.

tpgrpdel

See the description of the `tpgrpdel` command in the *BEA Tuxedo Reference manual*.

tpgrpmod

See the description of the `tpgrpmod` command in the *BEA Tuxedo Reference manual*.

tpusradd

See the description of the `tpusradd` command in the *BEA Tuxedo Reference manual*.

tpusrdel

See the description of the `tpusrdel` command in the *BEA Tuxedo Reference manual*.

tpusrmod

See the description of the `tpusrmod` command in the *BEA Tuxedo Reference manual*.

weblogic.rmc

Synopsis

A *proxy* is a class used by the clients of a remote object to handle the marshaling and unmarshaling of parameters across a network. In RMI, the stub and skeleton class files generated by the RMI compiler are *proxies* for the RMI client and RMI server objects, respectively.

The Weblogic RMI compiler (`weblogic.rmic`) is a tool for generating *stubs* for RMI clients and *skeletons* for RMI servers.

To generate stubs and skeletons, run the WebLogic RMI compiler on the fully-qualified package name of the compiled class that contains the remote object implementation. (Note that you must first have generated class files by running the `javac` compiler on the Java source files.)

Syntax

The syntax for using the WebLogic RMI compiler is as follows:

```
java weblogic.rmic [options] ClassName
```

Options

The options to the java `weblogic.rmic` command are shown in the following table.

Option	Description
`-help`	Prints the complete list of command line options.
`-version`	Prints version information.
`-d <dir>`	Indicates (top-level) directory for compilation.
`-notransactions`	Skip transaction context propagation
`-verbosemethods`	Instruments proxies to print debug information to `std err`.
`-descriptor <example>`	Associates or creates a descriptor for each remote class.
`-visualCafeDebugging`	Instruments proxies to support distributed debugging under VisualCafe.
`-v1.2`	Generates Java 1.2 style stubs
`-keepgenerated`	Keeps the generated `.java` files.
`-commentary`	Emit commentary.

Option	Description
`-compiler <JavaCompiler>`	Explicitly indicate which Java compiler to use. For example: `java weblogic.rmic -compiler sj examples.hello.HelloImpl`
`-g`	Compile debugging info into class file.
`-O`	Compile with optimization on.
`-debug`	Compile with debugging on.
`-nowarn`	Compile without warnings.
`-verbose`	Compile with verbose output.
`-nowrite`	Do not generate .class files.
`-deprecation`	Warn about deprecated calls.
`-normi`	Passed through to the Symantec sj compiler.
`-J<option>`	Flags passed through to java runtime.
`-classpath <path>`	Classpath to use during compilation.

The `weblogic.rmic` command also accepts any option supported by `javac`—the options are passed directly to the Java compiler.

Description

To create a proxy *stub* file for the client and *skeleton* file for the server, you must run the `weblogic.rmic` compiler on the fully-qualified package names of compiled class files that contain remote object implementations, like `my.package.MyImpl_WLstub`. The `weblogic.rmic` command takes one or more class names as an argument and produces class files of the form `MyImpl_WLStub.class` and `MyImpl_WLSkel.class`.

(Note that you must first have generated class files by running the `javac` compiler on the Java source files.)

For example, to generate the stub and skeleton class files for the class `classes/my/package/MyImpl.class`, you would change directories (`cd`) into the `classes` directory and run the `weblogic.rmic` command on the generated class file as follows:

```
java weblogic.rmic -d . my.package.MyImpl
```

The `weblogic.rmic` command accepts any option supported by `javac`—the options are passed directly to the Java compiler. In the example, the `-d` option indicates the root directory in which to place the compiled stub and skeleton class files. So the preceding command creates the following files in the directory `classes/my/package`:

```
MyImpl_WLStub.class
```

```
MyImpl_WLSkel.class
```

The generated stub class implements exactly the same set of remote interfaces as the remote object itself, and handles the necessary encoding (*marshaling*) and decoding (*unmarshaling*) of parameters sent across the network.

The skeleton class is also generated by the WebLogic RMI compiler but the skeleton is not used in WebLogic RMI. Generally, the RMI skeleton would unmarshal the invoked method and arguments on the remote object, invoke the method on the instance of the remote object, and then marshal the results for return to the client. WebLogic Enterprise handles the unmarshaling, method invocation, and marshalling on the RMI server side using reflection. If necessary, you can discard the generated skeleton class files to save disk space.

2 Server Process and File Format Reference

The WebLogic Enterprise system uses the following server processes and files:

- TMFFNAME
- TMIFRSVR
- factory_finder.ini
- UBBCONFIG

This topic describes these server processes.

TMFFNAME

Synopsis

Server that runs the FactoryFinder and supporting NameManager services.

Syntax

```
TMFFNAME SRVGRP="identifier" SRVID="number"
     [CLOPT="[-A] [servopts options]
[-- [-F ] [-N | -N –M [-f filename]]]"]
```

Description

`TMFFNAME` is a server provided by WebLogic Enterprise that runs the FactoryFinder and supporting NameManager services which maintain a mapping of application-supplied names to object references.

Parameters

`-A`
: Advertise all services built into the server

`-F`
: FactoryFinder service

`-N`
: Slave NameManager service
This is the default.

`–M`
: Master NameManager service

`-f filename`
: Location of FactoryFinder import file

The FactoryFinder service is a CORBA-derived service that provides client applications with the ability to find application factories that correspond to application-specified search criteria. Consult the *C++ Programming Reference* for a complete description on the FactoryFinder API and *Creating C++ Server Applications* for a description of registering and unregistering factories. The FactoryFinder service is the "default" service if no services are specified in the CLOPT.

The NameManager service is a WebLogic Enterprise-specific service that maintains a mapping of application-supplied names to object references. One usage of this service is to maintain the application factory name-to-object reference list. The NameManager service can be booted with an `-M` option that designates a Master role. If the `-M` option is not specified, the NameManager is assumed to be a Slave. Slave NameManagers obtain updates from the Master. Only one Master NameManager can be specified in an application.

The master NameManager can be configured to support the location of factory objects that reside in a remote domain through the use of an initialization file (for example `import_factories.ini`). The location of the initialization file is specified with the `-f` command-line option.

If the `-f` option is specified and the factoyfinder.ini file is not found, the initialization of the master NameManager will fail. If the `-f` option is not specified, the masterNameManger will be initialized however, the process will only contain locally registered factory objects.

Note: It is possible to boot one or more `TMFFNAME` processes running the same service. To provide increased reliability, at least two NameManager services must be configured, preferably on different machines.

Interoperability

The TMFFNAME servers run on WebLogic Enterprise version 4.0 software and later.

Notes

If there are less than two NameManager services configured in the application's `UBBCONFIG` (`TMFFNAME -N`), the server terminates itself during boot and writes an error message to the user log.

If a Master NameManager service is not configured in the application's `UBBCONFIG` file and is running when a Slave NameManager service starts, the server terminates itself during boot and writes an error message to the user log. Additionally, if the Master is down, registration and unregistration of factories is disabled until the Master restarts.

If a `TMSYSEVT` server is not configured in the application's `UBBCONFIG` file and is not running when a NameManager service is being started, the server terminates itself during boot and writes an error message to the user log.

If a NameManager service is not configured in the application's `UBBCONFIG` file and a FactoryFinder service is being started, the server terminates itself during boot and writes an error message to the user log.

Example

```
*SERVERS
TMSYSEVT SRVGRP=ADMIN1 SRVID=44 RESTART=Y
          CLOPT="-A"

TMFFNAME SRVGRP=ADMIN1 SRVID=45 RESTART=Y
CLOPT="-A -- -F"

TMFFNAME SRVGRP=ADMIN1 SRVID=46 RESTART=Y
CLOPT="-A -- -N –M –f c:\appdir\import_factories.ini"
TMFFNAME SRVGRP=ADMIN2 SRVID=47 RESTART=Y
CLOPT="-A -- -N"
```

```
TMFFNAME SRVGRP=ADMIN3 SRVID=48 RESTART=Y
CLOPT="-A -- -F"
TMFFNAME SRVGRP=ADMIN4 SRVID=49 RESTART=Y
CLOPT="-A -- -F"
```

See Also `factory_finder.ini`, `TMSYSEVT`(5), `userlog`(3), `UBBCONFIG`(5) in the *BEA Tuxedo Reference*, and the TP Framework chapter in the *C++ Programming Reference*.

TMIFRSVR

Synopsis The Interface Repository server.

Syntax

```
TMIFRSVR SRVGRP="identifier" SRVID="number" RESTART=Y GRACE=0
CLOPT="[ servopts options  ] -- [ -f repository_file_name ]"
```

Description The TMIFRSVR server is a server provided by BEA for accessing the Interface Repository. The API is a subset of the CORBA-defined Interface Repository API. For a description of the Interface Repository API, see the *C++ Programming Reference*.

Parameter

[-f *repository_file_name*]

Interface Repository file name. This file must have been generated previously using the idl2ir command. If this parameter is not specified, the default repository file name repository.ifr located in the application directory (APPDIR) for the machine is used. If the repository file cannot be read, the server fails to boot.

Examples

```
*SERVERS

#This server uses the default repository TMIFRSVR
SRVGRP="IFRGRP" SRVID=1000 RESTART=Y GRACE=0

#This server uses a non-default repository TMIFRSVR
SRVGRP="IFRGRP" SRVID=1001 RESTART=Y GRACE=0
CLOPT="-- -f /nfs/repository.ifr"
```

See Also ir2idl, UBBCONFIG(5), and servopts(5) in the *BEA Tuxedo Reference*.

factory_finder.ini

Name ASCII FactoryFinder domain configuration file

Description `factory_finder.ini` is the FactoryFinder configuration file for domains. This file is parsed by the `TMFFNAME` service when it is started as a Master NameManager. The file contains information used by NameManagers to control the import and the export of object references for factory objects with other domains. To use the information in the `factory_finder.ini` file, you must specify the `factory_finder.ini` file in the `-f` option of the TMFFNAME server process.

Definitions A WebLogic Enterprise system domain application is defined as the environment described in a single `TUXCONFIG` file. A WebLogic Enterprise system application can communicate with another WebLogic Enterprise system application or with another TP application via a domain gateway group. In "WebLogic Enterprise system domain" terms, an application is the same as a TP domain.

A Remote Factory is a factory object that exists in a remote domain that is made available to the application through a WebLogic Enterprise FactoryFinder.

A Local Factory is a factory object that exists in the local domain that is made available to remote domains through a WebLogic Enterprise FactoryFinder.

File Format The file is made up of two specification sections. Allowable section names are: `DM_REMOTE_FACTORIES` and `DM_LOCAL_FACTORIES`.

- Formatting Guidelines

  Parameters are generally specified by: `KEYWORD = value`. This sets `KEYWORD` to `value`. Valid keywords are described within each section. `KEYWORD`s are reserved; they cannot be used as values, unless they are quoted.

  If a value is an `identifier`, standard C rules are used. An `identifier` must start with an alphabetic character or underscore and must contain only alphanumeric characters or underscores. An `identifier` cannot be the same as any `KEYWORD`.

  A value that is not an `identifier` must be enclosed in double quotes.

  Input fields are separated by at least one space or tab character.

  "#" introduces a comment. A newline ends a comment.

  Blank lines and comments are ignored.

Lines are continued by placing at least one tab after the newline. Comments can not be continued.

- `DM_LOCAL_FACTORIES` section

  This section provides information about the factories exported by each local domain. This section is optional; if it is not specified, all local factory objects can be exported to remote domains. If this section is specified, it should be used to restrict the set of local factory objects that can be retrieved from a remote domain. The reserved `factory_id.factory_kind` identifier of "NONE" can be used to restrict any local factory from being retrieved from a remote domain.

  Lines within this section have the form:

  ```
  factory_id.factory_kind
  ```

  where `factory_id.factory_kind` is the local name (`identifier`) of the factory. This name must correspond to the identifier of a factory object registered by one or more WebLogic Enterprise server applications with the WebLogic Enterprise FactoryFinder.

  The `factory_kind` must be specified for `TMFFNAME` to locate the appropriate factory. An entry that does not contain a `factory_kind` value does not default to a value of `"FactoryInterface"`.

- `DM_REMOTE_FACTORIES` section

  This section provides information about factory objects "imported" and available on remote domains. Lines within this section have the form:

  ```
  factory_id.factory_kind required parameters
  ```

  where `factory_id.factory_kind` is the name (`identifier`) of the factory object used by the local WebLogic Enterprise system domain application for a particular remote factory object. Remote factory objects are associated with a particular remote domain.

Note: If you use the `TobjFactoryFinder` interface, the `factory_kind` must be `FactoryInterface`.

The required parameter is:

```
DOMAINID = domain_id
```

This parameter specifies the identity of the remote domain in which the factory object is to be retrieved. The `DOMAINID` must not be greater than 32 octets in length. If the value is a string, it must be 32 characters or fewer (counting the

trailing null). The value of `domain_id` can be a sequence of characters or a sequence of hexadecimal digits preceded by "0x".

The optional parameter is:

```
RNAME = string
```

This parameter specifies the name exported by remote domains. This value will be used by a remote domain to request this factory object. If this parameter is not specified, the remote factory object name is the same as the named specified in `factory_id.factory_kind`.

Multiple entries with the same name can be specified as long as the values associated with either the `DOMAINID` or `RNAME` parameter results in the identification of a unique factory object.

Examples

- **Example 1**

The following FactoryFinder domain configuration file defines two entries for a factory object that will be known in the local domain by the identifier `Teller.FactoryIdentity` that is imported from two different remote domains:

```
# BEA WebLogic Enterprise FactoryFinder Domain
# Configuration File
#
*DM_REMOTE_FACTORIES
  Teller.FactoryIdentity
    DOMAINID="Northwest"
    RNAME=Teller.FactoryType
  Teller.FactoryIdentity
     DOMAINID="Southwest"
```

In the first entry, a factory object is to be imported from the remote domain with an identity of "Northwest" that has been registered with a factory identity of `Teller.FactoryType`.

In the second entry, a factory object is to be imported from the remote domain with an identity of "Southwest" that has been registered with a factory identity of `Teller.FactoryIdentity`. Note that because no `RNAME` parameter was specified, the name of the factory object in the remote domain is assumed to be the same as the factory's name in the local domain.

- **Example 2**

The following FactoryFinder domain configuration file defines that only factory objects registered with the identity of `Teller.FactoryInterface` in the local

domain are allowed to be exported to any remote domain. Requests for any other factory should be denied.

```
# BEA WebLogic Enterprise FactoryFinder Domain
# Configuration File
#
*DM_LOCAL_FACTORIES
  Teller.FactoryInterface
```

- **Example 3**

  The following FactoryFinder domain configuration file defines that none of the factory objects registered with the WebLogic Enterprise FactoryFinder are to be exported to a remote domain.

```
# BEA WebLogic Enterprise FactoryFinder Domain
# Configuration File
#
*DM_LOCAL_FACTORIES
   NONE
```

UBBCONFIG

See the description of the `UBBCONFIG` in the *BEA Tuxedo Reference manual*.

3 MIB Reference

The following BEA Tuxedo management information bases (MIBS) have been added or enhanced for the WebLogic Enterprise product:

- T_IFQUEUE Class
- T_INTERFACE Class
- T_JDBCCONNPOOL Class
- T_ROUTING Class
- T_SERVER Class

This topic describes these MIBs.

T_IFQUEUE Class

Overview

The T_IFQUEUE MIB class represents runtime attributes of an interface as it pertains to a particular server queue (T_QUEUE). This is primarily a read-only class providing access to the inherited configuration attributes of an interface as well as statistics relating to the interface on the queue. Additionally, this class gives administrators finer granularity in suspending and activating interfaces. This class provides the link between an interface name and the server processes capable of processing method invocations on the interface, that is, TA_RQADDR can be used as a key search field on the T_SERVER class.

Attribute Table

T_IFQUEUE Class Definition Attribute Table

Attribute	Usage	Type	Permissions	Values	Default
TA_INTERFACENAME	*	string	R--R--R--	*string*[1..128]	N/A
TA_SRVGRP	*	string	R--R--R--	*string*[1..30]	N/A
TA_RQADDR	*	string	R--R--R--	*string*[1..30]	N/A
TA_STATE	k	string	R-XR-XR--	GET: "{ACT|SUS|PAR}" SET: "{ACT|SUS}"	N/A
TA_AUTOTRAN		string	R--R--R--	"{Y|N}"	N/A
TA_LOAD		long	R--R--R--	1 <= *num* < 32K	N/A
TA_PRIO		long	R--R--R--	1<= *num* < 101	N/A
TA_TIMEOUT		long	R--R--R--	0 <= *num*	N/A
TA_TRANTIME		long	R--R--R--	0 <= *num*	N/A
TA_FBROUTINGNAME		string	R--R--R--	*string*[1...15]	N/A
TA_LMID	k	string	R--R--R--	LMID	N/A
TA_NUMSERVERS		long	R--R--R--	0 <= *num*	N/A
TA_TPPOLICY		string	R--R--R--	"{method | transaction | process}"	N/A
TA_TXPOLICY		string	R-R-R--	"{always | never | optional | ignore}"	N/A
TA_NCOMPLETED	l	long	R-XR-XR--	0 <= *num*	N/A

TA_NQUEUED	l	long	R--R--R--	0 <= *num*	N/A
TA_CUROBJECTS	l	long	R--R--R--	0 <= *num*	N/A
TA_CURTRANSACTIONS	l	long	R--R--R--	0 <= *num*	N/A

(k) - GET key field
(l) - Local Field
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_INTERFACENAME: *string*[1..128]
: The fully qualified interface name. The interface repository id for the interface. The format of this name is dependent on the options specified in the IDL which generates the interface implementation. See CORBA 2.1 Specification Section 7.6 [CORBA] for details.

TA_SRVGRP: *string*[0..30]
: Server group name. Server group names cannot contain an asterisk, comma or colon.

TA_RQADDR: *string*[1..30]
: Symbolic address of the request queue for an active server offering this interface. See T_SERVER:TA_RQADDR for more information on this attribute.

TA_STATE:

GET: {ACTive | SUSpended | PARtitioned}
: A GET operation will retrieve configuration information for the selected T_IFQUEUE objects. The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_IFQUEUE object represents an available interface in the running system.
SUSpended	T_IFQUEUE object represents a currently suspended interface in the running system.
PARtitioned	T_IFQUEUE object represents a currently partitioned interface in the running system.

SET: { ACTive | SUSpended}
: The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

ACTive	Activate the T_IFQUEUE object. State change only allowed when in the SUSpended state. Successful return leaves object in ACTive state.
SUSpended	Suspend the T_IFQUEUE object. State change only allowed when in the ACTive state. Successful return leaves object in SUSpended state.

Limitation: Dynamic advertisement of interfaces (i.e., state change from INActive or INValid to ACTive) is not supported, nor is unadvertisement (i.e., state change from ACTive to INActive).

TA_AUTOTRAN: { Y | N }
: Signifies whether a transaction will be automatically started for invocations made outside a transaction context. See T_INTERFACE description of this attribute for discussion of limitations regarding this attribute.

TA_LOAD: 1 <= num <= 32K
: This T_INTERFACE object imposes the indicated load on the system. Interface loads are used for load balancing purposes, that is, queues with higher enqueued workloads are less likely to be chosen for a new request.

TA_PRIO: 1 <= num <= 101
: This T_INTERFACE object has the indicated dequeuing priority. If multiple interface requests are waiting on a queue for servicing, the higher priority requests will be handled first.

TA_TIMEOUT: 0 <= num
: Time limit (in seconds) for processing individual method invocations for this interface. Servers processing method invocations for this interface will be abortively terminated if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the server should not be abortively terminated.

TA_TRANTIME: 0 <= num
: Transaction timeout value in seconds for transactions automatically started for this T_INTERFACE object. Transactions are started automatically when a requests not in transaction mode is received and the T_INTERFACE:TA_AUTOTRAN attribute value for the interface is "Y".

TA_FBROUTINGNAME: string[1..15]
: The factory-based routing criteria associated with this interface.

TA_LMID: LMID
: Current logical machine on which the queue offering this interface is located.

TA_NUMSERVERS: 0 <= num
: Number of corresponding servers offering this interface on this queue.

TA_TPPOLICY: { method | transaction | process }
: The TP framework deactivation policy. This reflects the policy registered with the framework at server startup. The first server to register the interface sets the value in T_INTERFACE. This value cannot be changed.

TA_TXPOLICY: { optional | always | never | ignore }
: The transaction policy for the interface. The setting in this attribute affects the effect of the TA_AUTOTRAN attribute. See TA_AUTOTRAN for further explanation. This attribute is always read-only. It is set by the developer when the server is built and registered at server startup.

TA_NCOMPLETED: 0 <= num
: Number of interface method invocations completed since the interface was initially offered.

TA_NQUEUED: 0 <= num
: Number of requests currently enqueued for this interface.

TA_CUROBJECTS: 0 <= num
: Number of active objects for this interface for associated queue. This number represents the number of entries in the active object table for this queue on the associated machine. This includes objects that are not in memory but that were invoked within an active transaction.

TA_CURTRANSACTIONS: 0 <= num
: Number of active global transactions associated with this interface for its associated queue.

T_INTERFACE Class

Overview

The T_INTERFACE MIB class represents configuration and runtime attributes of CORBA interfaces at both the domain and server group levels.

A domain-level T_INTERFACE object is one that is not associated with a Server Group. Its TA_SRVGRP attribute contains a null string (string of length 0, "").

A server group level T_INTERFACE object is one that has an associated server group (i.e., its TA_SRVGRP attribute contains a valid server group name for the domain). This Server Group level representation of an interface also provides a container for managing interface state (TA_STATE) and for collecting accumulated statistics.

An associated server group level T_INTERFACE object must exist for any CORBA Interfaces that are activated in a server. The activation of interfaces in a server is controlled by the state of a T_IFQUEUE object for the interface. Activation of a T_IFQUEUE object causes its attributes to be initialized with the values specified for the associated server group level T_INTERFACE object. If such an object does not exist, then one will be dynamically created. This dynamically-created server group level T_INTERFACE object will be initialized with the attributes of the domain level T_INTERFACE object for the interface if one exists. If an associated domain level T_INTERFACE object does not exist, system specified default configuration values will be applied. Once activated, interfaces are always associated with a server group level T_INTERFACE object.

The specification of configuration attributes for interfaces at any level is completely optional, system defined defaults will be provided and run-time server group level T_INTERFACE objects will be created. Interfaces to be offered by a server are identified via the ICF file used to generate server skeletons and advertised automatically by the system at server activation time.

Attribute Table

T_INTERFACE Class Definition Attribute Table

Attribute	Usage	Type	Permissions	Values	Default
TA_INTERFACENAME	r*	string	ru-r--r--	*string*[1..128]	N/A
TA_SRVGRP	r*	string	ru-r--r--	*string*[0..30]	N/A
TA_STATE	k	string	rwxr-xr--	GET: "{ACT | INA | SUS | PAR}" SET: "{NEW | INV | ACT | REA | SUS}"	N/A
TA_AUTOTRAN		string	rwxr-xr--	"{Y|N}"	"N"[a]
TA_LOAD		long	rwxr-xr--	1 <= *num* < 32K	50[1]
TA_PRIO		long	rwxr-xr--	1<= *num* < 101	50[a]
TA_TIMEOUT		long	rwxr-xr--	0 <= *num*	0[a]
TA_TRANTIME		long	rwxr-xr--	0 <= *num*	30[a]
TA_FBROUTINGNAME		string	rwyr-yr--	string[1...15]	[2]
TA_LMID	k	string	R--R--R--	LMID	N/A
TA_NUMSERVERS		long	R--R--R--	0 <= *num*	N/A
TA_TPPOLICY		string	R--R--R--	"{method | transaction | process}"	N/A
TA_TXPOLICY		string	R-R-R--	"{always | never | optional | ignore}"	N/A
TA_NCOMPLETED	l	long	R-XR-XR--	0 <= *num*	N/A[3]
TA_NQUEUED	l	long	R--R--R--	0 <= **num**	N/A

(k) - GET key field
(l) - Local Field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

1. Group level T_INTERFACE objects (TA_SRVGRP != "") determine their defaults from the domain level T_INTERFACE object with a matching TA_INTERFACENAME setting if one exists. The

2. All T_INTERFACE objects with the same TA_INTERFACENAME must have matching TA_FBROUTINGNAME values. Therefore, the default for a newly configured object is the 0 length string ("") if there are currently no matching objects with the same TA_INTERFACENAME. Otherwise, the default (and in fact only legal value) is the currently configured TA_FBROUTINGNAME

3. TA_NCOMPLETED and TA_IMPLID (locals) require TA_LDBAL="Y" in the T_DOMAIN MIB class.

Attribute Semantics

TA_INTERFACENAME: string[1..128]
: The fully qualified interface name. The interface repository id for the interface. The format of this name is dependent on the options specified in the IDL which generates the interface implementation. See CORBA 2.1 Specification Section 7.6 [CORBA] for details.

TA_SRVGRP: string[0..30]
: Server group name. Server group names cannot contain an asterisk, comma or colon. An explicitly specified 0 length string for this attribute is used to specify and query domain level configuration and runtime information for an interface. There are certain limitations and semantic differences noted in other attributes with respect to domain and group level objects in this class.

TA_STATE:
: Following are the semantics for GET and SET TA_STATE values on the T_INTERFACE class. Where semantics differ between group and domain level objects, those differences are noted.

GET: {ACTive | INActive | SUSpended | PARtitioned}
: A GET operation will retrieve configuration information for the selected T_INTERFACE objects. The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_INTERFACE object is defined and at least one corresponding T_IFQUEUE entry is in the ACTive state. **Note:** For a group level T_INTERFACE object, corresponding T_IFQUEUE entries are those with matching TA_INTERFACENAME and TA_SRVGRP attributes. For a domain level T_INTERFACE object, corresponding T_IFQUEUE entries are those with matching TA_INTERFACENAME attributes regardless of their TA_SRVGRP value.
INActive	T_INTERFACE object is defined and there are no corresponding T_IFQUEUE entries in any ACTive equivalent state.
SUSpended	T_INTERFACE object is defined and amongst all corresponding T_IFQUEUE entries there are none in the ACTive state and at least one in the SUSpended state. This state is ACTive equivalent for the purpose of determining permissions.
PARtitioned	T_INTERFACE object is defined and amongst all corresponding T_IFQUEUE entries there are 1. none in the ACTive state 2. none in the SUSpended state and 3. at least one in the PARtitioned state. This state is ACTive equivalent for the purpose of determining permissions.

SET: {NEW | INValid | ACTive | REActivate | SUSpended}

A SET operation will update configuration and runtime information for the selected T_INTERFACE object. Note that modifications may affect more than one server group when making domain level changes and runtime modifications may affect more than one server if multiple servers are currently offering an interface. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_INTERFACE object for application. State change only allowed when in the INValid state. Successful return leaves object in INActive state. Creation of a domain level T_INTERFACE object will affect existing group level objects with the same TA_INTERFACENAME value by resetting all TA_FBROUTINGNAME values if a new value is explicitly specified. All other configuration attribute settings will not affect existing group level T_INTERFACE objects.

INValid	Delete T_INTERFACE object for application. State change only allowed when in the INActive state. Successful return leaves object in INValid state.
ACTive	Activate the T_INTERFACE object. Setting this state on the domain level object has the effect of activating all corresponding T_IFQUEUE entries that are currently SUSpended throughout the domain. Setting this state on the group level object will affect only servers within the group offering the interface. State change only allowed when in the SUSpended state. Successful return leaves object in ACTive state.
REActivate	Reactivate the T_INTERFACE object. Setting this state on the domain level object has the effect of activating all corresponding T_IFQUEUE entries that are currently SUSpended throughout the domain. Setting this state on the group level object will affect only servers within the group offering the interface. State change only allowed when in the ACTive or SUSpended states. Successful return leaves object in ACTive state. This state permits global activation of T_IFQUEUE entries suspended at the group level without having to individually activate each group level T_INTERFACE object.
SUSpended	Suspend the T_INTERFACE object. Setting this state on the domain level object has the effect of suspending all corresponding T_IFQUEUE entries that are currently ACTive throughout the domain. Setting this state on the group level object will affect only servers within the group offering the interface. State change only allowed when in the ACTive state. Successful return leaves object in SUSpended state.

Limitation: Dynamic advertisement of interfaces (i.e., state change from INActive or INValid to ACTive) is not supported, nor is unadvertisement (i.e., state change from ACTive to INActive).

TA_AUTOTRAN: { Y | N }

Signifies whether a transaction will be automatically started for invocations made outside a transaction context.

Limitations: Run-time updates to this attribute are not reflected in active equivalent T_INTERFACE objects and TA_TXPOLICY may override the value specified for this attribute in the ubbconfig file. If TA_TXPOLICY is:

always	A value of N will have no effect at run time. Behavior will be as though the setting was Y.
never	A value of Y will have no effect. The interface will never be involved in a transaction.

ignore	A value of Y will have no effect. The interface will never be involved in a transaction.

TA_LOAD: 1 <= *num* <= 32K

This T_INTERFACE object imposes the indicated load on the system. Interface loads are used for load balancing purposes, that is, queues with higher enqueued workloads are less likely to be chosen for a new request.
Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

TA_PRIO: 1 <= *num* <= 101

This T_INTERFACE object has the indicated dequeuing priority. If multiple interface requests are waiting on a queue for servicing, the higher priority requests will be handled first.
Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

TA_TIMEOUT: 0 <= *num*

Time limit (in seconds) for processing individual method invocations for this interface. Servers processing method invocations for this interface will be abortively terminated if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the server should not be abortively terminated.
Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

TA_TRANTIME: 0 <= *num*

Transaction timeout value in seconds for transactions automatically started for this T_INTERFACE object. Transactions are started automatically when a requests not in transaction mode is received and the T_INTERFACE: TA_AUTOTRAN attribute value for the interface is "Y".
Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.
Note: Updating this value at runtime for domain level objects should cause a warning, since the only use would be to set the default for a subsequent boot of the application.

TA_FBROUTINGNAME: *string*[1..15]

The factory-based routing criteria associated with this interface. The name FBROUTINGNAME is used to allow for the future possibility of other routing criteria for message-based routing. This will be less confusing than trying to overload ROUTINGNAME

Limitation: This attribute may be set only for a domain level T_INTERFACE object, i.e., TA_SRVGRP is "".

TA_LMID: *LMID*
: Current logical machine with which the active equivalent group level T_INTERFACE object is associated. This attribute is blank, i.e., "" for domain level objects unless a local query is performed, i.e., TA_FLAGS has the MIB_LOCAL bit set. In the local case, multiple domain level objects will be returned for the same interface, one per machine, with the local values retrieved from each machine represented in the separate objects.

TA_NUMSERVERS: 0 <= *num*
: Number of corresponding servers offering this interface.

TA_TPPOLICY: { method | transaction | process }
: The TP framework deactivation policy. This reflects the policy registered with the framework at server startup. The first server to register the interface sets the value in T_INTERFACE. This value cannot be changed.

TA_TXPOLICY: { optional | always | never | ignore }
: The transaction policy for the interface. The setting in this attribute affects the effect of the TA_AUTOTRAN attribute. See TA_AUTOTRAN for further explanation. This attribute is always read-only. It is set by the developer when the server is built and registered at server startup.

TA_NCOMPLETED: 0 <= *num*
: Number of interface method invocations completed with respect to the corresponding T_IFQUEUE objects since they were initially offered. Local queries (TA_FLAGS MIB_LOCAL bit set) on domain level objects will return one object per machine with the statistics for the indicated interface on that machine.

TA_NQUEUED: 0 <= *num*
: Number of requests currently enqueued for this interface. Local queries (TA_FLAGS MIB_LOCAL bit set) on domain level objects will return one object per machine with the statistics for the indicated interface on that machine.

Implementation Hint

The T_INTERFACE MIB is a mapping from an interface to a BEA Tuxedo service. The MIB server can implement some of the get/set operations for an interface by calling the existing logic for the associated T_SERVICE object.

T_JDBCCONNPOOL Class

Overview

This class represents the configuration and runtime attributes of JDBC connection pools on a Java server. The attributes consist of statistics or values associated with each connection pool. Except for TA_STATE, attribute values are persistent in TUXCONFIG. Local attributes are local to the memory allocated to a Java server.

Attribute Table

T_JDBCCONNPOOL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVID(r)(*)	string	ru-r--r--	1<=*num*<30,001	N/A
TA_SRVGRP(r)(*)	string	ru-r--r--	*string[1...30]*	N/A
TA_DSNAME(r)(*)	string	ru-r--r--	*string[0...30]*	N/A
TA_DRIVER	string	rw-r--r--	*string[0...256]*	N/A
TA_URL	string	rw-r--r--	*string[2...256]*	N/A
TA_STATE	string	rw-r--r--	GET:VALID SET:"{NEW|INVALID}"	N/A N/A
TA_DBNAME	string	rw-r--r--	*string*[0...30]	N/A
TA_DBUSER	string	rw-r--r--	*string*[0...30]	N/A
TA_DBPASSWORD	string	rw------	*string[0...64]*	N/A
TA_USERROLE	string	rw-r--r--	*string*[0...30]	N/A
TA_DBHOST	string	rw-r--r--	*string[0...30]*	N/A
TA_DBNETPROTOCOL	string	rw-r--r--	*string[0...30]*	N/A
TA_DBPORT	long	rw-r--r--	0<*num*<64K	N/A
TA_PROPS	string	rw-r--r--	*string*[0...256]	N/A
TA_ENABLEXA	string	rw-r--r--	{Y | N}	N
TA_CREATEONSTARTUP	string	rw-r--r--	{Y | N}	Y
TA_LOGINDELAY	long	rw-r--r--	0<=*num*	0
TA_INITCAPACITY	long	rw-r--r--	*0*<*num*	N/A
TA_MAXCAPACITY	long	rw-r--r--	*0*<*num*	N/A
TA_CAPACITYINCR	long	rw-r--r--	*0*<*num*	N/A

T_JDBCCONNPOOL Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_ALLOWSHRINKING	string	rw-r--r--	{Y | N}	N
TA_SHRINKPERIOD	long	rw-r--r--	*1<=num*	15
TA_TESTTABLE	string	rw-r--r--	*string[0...256]*	N/A
TA_REFRESH	long	rw-r--r--	*0<=num*	5
TA_TESTONRESERVE	string	rw-r--r--	{Y | N}	N
TA_TESTONRELEASE	string	rw-r--r--	{Y | N}	N
TA_WAITFORCONN	string	rw-r--r--	{Y | N}	Y
TA_WAITTIMEOUT	long	rw-r--r--	0<=*num*	N/A
T_JDBCCONNPOOL Class:LOCAL Attributes				
TA_CONNUSED	long	R--R--R--	0<=*num*	N/A
TA_CONNAVAILABLE	long	R--R--R--	0<=*num*	N/A
TA_HWMCONNUSE	long	R--R--R--	0<=*num*	N/A
TA_HWMCONNCREATED	long	R--R--R--	0<=*num*	N/A
TA_AWAITINGCONN	long	R--R--R--	0<=*num*	N/A
TA_HWMFORWAIT	long	R--R--R--	0<=*num*	N/A

(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_SRVID 1<=num<30,001
: Together with the server group name, this value is used to identify a Java server, specified in the SERVERS section of the UBBCONFIG, for which the connection pool is being described.

TA_SRVGRP *string*[1...30]
: Name of a server group. This is used to identify a Java server, specified in the SERVERS section of the UBBCONFIG, for which the connection pool is being described.

TA_DSNAME *string*[0...30]
: The data source name for the connection pool.

TA_DRIVER *string[0...256]*
: The class name for the Java driver.

TA_URL *string*[0...256]
URL for a JDBC driver that is not JDBC 2.0-compliant.

TA_STATE
The INVALID state is used to delete entries from the configuration file on a SET request. VALID is always returned by a GET request.

TA_DBNAME *string*[0...30]
The database name.

TA_DBUSER *string*[0...30]
User's account name.

TA_DBPASSWORD *string*[0...64]
The user's password. The password entered by the user should not exceed 24 bytes.

TA_DBUSERROLE *string*[0...30]
The user's SQL role.

TA_DBHOST *string*[0...30]
Database server name.

TA_DBNETPROTOCOL *string*[0...30]
The protocol used to communicate with the database.

TA_DBPORT 0<*num*<64K
The port used for database connections.

TA_PROPS *string*[0...256]
Vendor-specific information for the JDBC driver.

TA_ENABLEXA Y or N
If set to Y, indicates that the pool supports XA mode.

TA_CREATEONSTARTUP Y or N
If set to Y, indicates that the connection pool is created when the server is started. If set to N, the pool is created when the first request arrives.

TA_LOGINDELAY
The login delay in seconds.

TA_INITCAPACITY 0<*num*
The number of connections initially supported in the connection pool. *num* should not exceed the value of TA_MAXCAPACITY.

`TA_MAXCAPACITY` 0<*num*
: The maximum number of connections supported in the connection pool.

`TA_CAPACITYINCR` 0<*num*
: The number of connections added to the pool when the current limit is exceeded but the maximum capacity has not yet been reached.

`TA_ALLOWSHRINKING` `Y` or `N`
: If set to `Y`, allows connection pool shrinking.

`TA_SHRINKPERIOD 1=<`*num*
: The interval after which shrinking occurs, in minutes.

`TA_TESTTABLE` *string*[0...256]
: The name of a table in the database that is used to test the viability of connections in the connection pool. The query `select count(*) from TESTTABLE` is used to test a connection. The table must exist and be accessible to the database user for the connection.

`TA_REFRESH` *0<=num*
: The refresh interval, in minutes.

`TA_TESTONRESERVE` `Y` or `N`
: If set to `Y`, the Java server tests a connection after removing it from the pool and before giving it to the client. The test adds a small delay in serving the client's request for a connection from the pool but ensures that the client receives a working connection. A value for `TA_TESTTABLE` must be set for this feature to work.

`TA_TESTONRELEASE` `Y` or `N`
: If set to `Y`, the Java server tests a connection before returning it to the connection pool. If all the connections in the pool are already in use and a client is waiting for a connection, the client's wait will be slightly longer due to the test of the connection. A value for `TA_TESTTABLE` must be set for this feature to work.

`TA_WAITFORCONN` `Y` or `N`
: If set to `Y`, enables an application to wait for a connection indefinitely if none is currently available. If set to `N`, a request for a connection returns to the caller immediately if there is no connection available. `Y` is assumed unless `TA_WAITTIMEOUT` is specified, in which case it becomes `N`.

`TA_WAITTIMEOUT` 0<*num*

Time in seconds that an application will wait for a connection to become available.

T_ROUTING Class

Overview

The T_ROUTING class represents configuration attributes of routing specifications for an application. These attribute values identify and characterize application data dependent routing criteria with respect to field names, buffer types, and routing definitions.

Attribute Table

T_ROUTING Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_ROUTINGNAME(r)(*)	string	ru-r--r--	*string[1...15]*	N/A
TA_ROUTINGTYPE (r)	string	ru-r--r--	SERVICE or FACTORY	SERVICE
TA_BUFTYPE(r)(*)	string	ru-r--r--	*string[1...256]*	N/A[1]
TA_FIELD(r)(k) (*)	string	ru-r--r--	*string[1...30]*	N/A[1]
TA_RANGES(r)	carray	rw-r--r--	*carray[1...2048]*	N/A
TA_TYPE	string	ru-r--r--	string[1..15]	“SERVICE2
TA_FIELDTYPE (r)	string	rw-r--r--	string[1..30]	N/A
TA_STATE(k)	string	rw-r--r--	GET:"{VAL}" SET:"{NEW|INV}"	N/A N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

[1] TA_BUFTYPE only applies to BEA Tuxedo data-dependent routing criteria. TA_FIELDTYPE only applies to WebLogic Enterprise Factory-Based routing criteria. The specified u (uniqueness) permission applies only in the relevant case. That is: the combination of TA_ROUTINGNAME, TA_TYPE and TA_BUFTYPE must be unique for TA_TYPE=SERVICE, and TA_ROUTINGNAME, TA_TYPE and TA_FIELD must be unique for TA_TYPE=FACTORY.

The TA_TYPE attribute determines the permissible attributes for the TA_ROUTING object. TYPE=SERVICE corresponds to BEA Tuxedo data-dependent routing criteria. TYPE=FACTORY corresponds to WebLogic Enterprise factory-based routing. The default is SERVICE. SET operations are assumed to be for data-dependent routing if no TA_TYPE is specified. Specification of TA_FIELDTYPE is invalid for data-dependent routing. Specification of TA_BUFTYPE is invalid for factory-based routing.

Attribute Semantics

TA_ROUTINGNAME: *string[1...15]*

Routing criteria name.

TA_ROUTINGTYPE: *type*

Specifies the routing type. The default is TYPE=SERVICE to ensure that existing UBBCONFIG files used in BEA Tuxedo environments continue to work properly. Use TYPE=FACTORY if you are implementing factory-based routing for a WebLogic Enterprise interface.

TA_BUFTYPE: *type1*[*:subtype1[,subtype2* . . .]][*;type2*[*:subtype3*[, . . .]]] . . .

List of types and subtypes of data buffers for which this routing entry is valid. A maximum of 32 type/subtype combinations are allowed. The types are restricted to be one of FML, VIEW, X_C_TYPE, or X_COMMON. No subtype can be specified for type FML, and subtypes are required for types VIEW, X_C_TYPE, and X_COMMON (''*" is not allowed). Note that subtype names should not contain semicolon, colon, comma, or asterisk characters. Duplicate type/subtype pairs can not be specified for the same routing criteria name; more than one routing entry can have the same criteria name as long as the type/subtype pairs are unique. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same.

TA_FIELD: *string[1...30]*

The routing field name. When TA_TYPE=FACTORY, this is assumed to be a field that is specified in an NVList parameter to PortableServer::POA::create _reference_with_criteria for an interface that has this factory routing criteria associated with it. See section on factory-based routing for more details. When TA_TYPE=SERVICE this field is assumed to be an FML buffer or view field name that is identified in an FML field table (using the FLDTBLDIR and FIELDTBLS environment variables) or an FML view table (using the VIEWDIR and VIEWFILES environment variables), respectively. This information is used to get the associated field value for data dependent routing during the sending of a message.

TA_FIELDTYPE (Factory-based Routing Only)

Routing field type. This field is only valid if TA_TYPE=FACTORY. Valid types are: SHORT, LONG, FLOAT, DOUBLE, CHAR, STRING. Specification of this attribute is only valid for factory-based routing criteria.

TA_RANGES: *carray[1...2048]*

The ranges and associated server groups for the routing field. The format of *string* is a comma-separated, ordered list of range/group name pairs. A range/group name pair has the following format:

lower[-upper]:group

lower and *upper* are signed numeric values or character strings in single quotes. *lower* must be less than or equal to *upper*. To embed a single quote in a character string value, it must be preceded by two backslashes (for example, 'O\\'Brien'). The value MIN can be used to indicate the minimum value for the data type of the associated field on the machine. The value MAX can be used to indicate the maximum value for the data type of the associated field on the machine. Thus, "MIN--5" is all numbers less than or equal to -5, and "6-MAX" is all numbers greater than or equal to 6.

The meta-character ''*" (wildcard) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wild-card range is allowed per entry and it should be last (ranges following it will be ignored).

The routing field can be of any data type supported in FML. A numeric routing field must have numeric range values, and a string routing field must have string range values.

String range values for string, carray, and character field types must be placed inside a pair of single quotes and can not be preceded by a sign. Short and long integer values are a string of digits, optionally preceded by a plus or minus sign. Floating point numbers are of the form accepted by the C compiler or atof(3): an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

The group name indicates the associated group to which the request is routed if the field matches the range. A group name of ''*" indicates that the request can go to any group where a server offers the desired service.

Limitation: Attribute values greater than 256 bytes in length will disable interoperability with BEA Tuxedo Release 4.2.2 and earlier.

TA_STATE:

GET: {VALid}

A GET operation will retrieve configuration information for the selected T_ROUTING object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_ROUTING object is defined. Note that this is the only valid state for this class. Routing criteria are never ACTive; rather, they are associated through the configuration with service names and are acted upon at runtime to provide data dependent routing. This state is INActive equivalent for the purpose of permissions checking.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_ROUTING object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_ROUTING object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_ROUTING object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_ROUTING object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.

TA_TYPE

Routing criteria type. Valid values are "FACTORY" or "SERVICE". "FACTORY" specifies that the routing criteria applies to factory-based routing for a CORBA interface. The specification of TYPE=FACTORY is mandatory for a factory-based routing criteria. "SERVICE" specifies that the routing criteria applies to data-dependent routing for a BEA Tuxedo service. Default is "SERVICE". Specification of this attribute is optional for data-dependent routing criteria. Note that the type specified affects the validity and possible

values for other fields defined for this MIB class. These are noted for each field. TA_TYPE is required for SET operations for factory-based routing criteria.

Limitations None.

T_SERVER Class

Overview

The T_SERVER class represents configuration and run-time attributes of servers within an application. These attribute values identify and characterize configured servers as well as provide run-time tracking of statistics and resources associated with each server object. Information returned will always include fields that are common among all contexts of a server. In addition, for those servers that are not defined to the system as multicontexted (that is, those for which the value of TA_MAXDISPATCHTHREADS is 1), this class includes information about the server's context. For those servers that are defined to the system as multicontexted, placeholder values are reported for per-context attributes. Per-context attributes can always be found as part of the T_SERVERCTXT class. The T_SERVERCTXT class is defined even for single-contexted servers.

The TA_CLTLMID, TA_CLTPID, TA_CLTREPLY, TA_CMTRET, TA_CURCONV, TA_CURREQ, TA_CURRSERVICE, TA_LASTGRP, TA_SVCTIMEOUT, TA_TIMELEFT, and TA_TRANLEV attributes are specific to each server dispatch context. All other attributes are common to all server dispatch contexts.

TA_CLASSPATH, TA_JAVAHEAP, TA_JAVAHEAPUSE, TA_JAVAVERSION, and TA_JAVAVENDOR apply to Java servers only.

Attribute Table

T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	ru-r--r--	*string[1...30]*	N/A
TA_SRVID(r)(*)	long	ru-r--r--	1 <= *num* < 30,001	N/A
TA_SERVERNAME(k)(r)	string	rw-r--r--	*string[1...78]*	N/A
TA_SRVTYPE	string	r--r--r--	JAVA	N/A
TA_GRPNO(k)	long	r--r--r--	1 <= *num* < 30,000	N/A
TA_STATE(k)	string	rwxr-xr--	GET:"{ACT|INA|MIG|CLE|RES|SUS|PAR|DEA}" SET:"{NEW|INV|ACT|INA|DEA}"	N/A N/A
TA_BASESRVID	long	r--r--r--	1 <= *num* < 30,001	N/A
TA_CLOPT	string	rwyr--r--	*string[0...256]*	"-A"
TA_ENVFILE	string	rwyr--r--	*string[0...78]*	""

T_SERVER Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_GRACE	long	rwyr--r--	0 <= *num*	86,400
TA_MAXGEN	long	rwyr--r--	1 <= *num* < 256	1
TA_MAX	long	rwxr--r--	1 <= *num* < 1,001	1
TA_MIN	long	rwyr--r--	1 <= *num* < 1,001	1
TA_RCMD	string	rwyr--r--	*string[0...78]*	""
TA_RESTART	string	rwyr--r--	"{Y|N}"	N
TA_SEQUENCE(k)	long	rwxr--r--	1 <= *num* < 10,000	>= 10,000
TA_SYSTEM_ACCESS	string	rwyr--r--	"{FASTPATH|PROTECTED}"	([1])
TA_CONV(k)	string	rw-r--r--	"{Y|N}"	N
TA_REPLYQ	string	rw-r--r--	"{Y|N}"	N
TA_RPPERM	long	rw-r--r--	0001 <= *num* <= 0777	([1])
TA_RQADDR(k)	string	rw-r--r--	*string[0...30]*	"*GRPNO.SRVID*"
TA_RQPERM	long	rw-r--r--	0001 <= *num* <= 0777	([1])
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_GENERATION	long	R--R--R--	1 <= *num* < 32K	N/A
TA_PID(k)	long	R--R--R--	1 <= *num*	N/A
TA_RPID	long	R--R--R--	1 <= *num*	N/A
TA_RQID	long	R--R--R--	1 <= *num*	N/A
TA_TIMERESTART	long	R--R--R--	1 <= *num*	N/A
TA_TIMESTART	long	R--R--R--	1 <= *num*	N/A
T_SERVER Class: LOCAL Attributes				
TA_CLASSPATH	string	R--R--R--	*string[0...2,047]*	N/A
TA_JAVAHEAP	long	R--R--R--	0<*num*	
TA_JAVAHEAPUSE	long	R--R--R--	1<=*num*<=100	N/A
TA_JAVAVENDOR	string	R--R--R--	*string[0...30]*	N/A
TA_JAVAVERSION	string	R--R--R--	*string[0...30]*	N/A
TA_MAXDISPATCHTHREADS	long	R--R--R--	0<*num*	N/A
TA_NUMCONV	long	R-XR-XR--	0 <= *num*	N/A

T_SERVER Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_NUMDEQUEUE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMENQUEUE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMPOST	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMREQ	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMSUBSCRIBE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRAN	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRANABT	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRANCMT	long	R-XR-XR--	0 <= *num*	N/A
TA_THREADSTACKSIZE	long	R--R--R--	0<*num*	N/A
TA_TOTREQC	long	R-XR-XR--	0 <= *num*	N/A
TA_TOTWORKL	long	R-XR-XR--	0 <= *num*	N/A
TA_CLTLMID	string	R--R--R--	*LMID*	N/A
TA_CLTPID	long	R--R--R--	1 <= *num*	N/A
TA_CLTREPLY	string	R--R--R--	"{Y|N}"	N/A
TA_CMTRET	string	R--R--R--	"{COMPLETE|LOGGED}"	N/A
TA_CURCONV	long	R--R--R--	0 <= *num*	N/A
TA_CURDISPATCHTHREADS	long	R--R--R--	*0*<*num*	N/A
TA_CUROBJECTS	long	R--R--R--	0 <= *num*	N/A
TA_CURINTERFACE	string	R--R--R--	*string*[0..128]	N/A
TA_CURREQ	long	R--R--R--	0 <= *num*	N/A
TA_CURRSERVICE	string	R--R--R--	*string[0...15]*	N/A
TA_CURTIME	long	R--R--R--	1 <= *num*	N/A
TA_HWDISPATCHTHREADS	long	R--R--R--	0<*num*	N/A
TA_LASTGRP	long	R--R--R--	1 <= *num* < 30,000	N/A
TA_SVCTIMEOUT	long	R--R--R--	0 <= *num*	N/A
TA_TIMELEFT	long	R--R--R--	0 <= *num*	N/A
TA_TRANLEV	long	R--R--R--	0 <= *num*	N/A

T_SERVER Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
(k) - GET key field (r) - Required field for object creation (SET TA_STATE NEW) (*) - GET/SET key, one or more required for SET operations				

[1]Defaults to value set for this attribute in Class T_DOMAIN

Attribute Semantics

TA_SRVGRP: *string[1...30]*

Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.

TA_SRVID: 1 <= *num* < 30,001

Unique (within the server group) server identification number.

TA_SERVERNAME: *string[1...78]*

Name of the server executable file. The server identified by TA_SERVERNAME will run on the machine(s) identified by the T_GROUP:TA_LMID attribute for this server's server group. If a relative pathname is given, then the search for the executable file is done first in TA_APPDIR, then in TA_TUXDIR/bin, then in /bin and /usr/bin, and then in <path>, where <path> is the value of the first PATH= line appearing in the machine environment file, if one exists. Note that the attribute value returned for an active server will always be a full pathname. The values for TA_APPDIR and TA_TUXDIR are taken from the appropriate T_MACHINE object. See discussion of the T_MACHINE:TA_ENVFILE attribute for a more detailed discussion of how environment variables are handled.

TA_GRPNO: 1 <= *num* < 30,000

Group number associated with this server's group.

TA_STATE:

GET:{ACTive|INActive|MIGrating|CLEaning|REStarting|SUSpended|PARtitioned|DEAd}

A GET operation will retrieve configuration and run-time information for the selected T_SERVER object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_SERVER object defined and active. This is not an indication of whether the server is idle or busy. An active server with a non 0-length TA_CURRSERVICE attribute should be interpreted as a busy server, that is, one that is processing a service request.
INActive	T_SERVER object defined and inactive.
MIGrating	T_SERVER object defined and currently in a state of migration to the server group's secondary logical machine. The secondary logical machine is the one listed in T_GROUP:TA_LMID attribute that does not match the T_GROUP:TA_CURLMID attribute. This state is ACTive equivalent for the purpose of determining permissions.
CLEaning	T_SERVER object defined and currently being cleaned up by the system after an abnormal death. Note that restartable servers may enter this state if they exceed TA_MAXGEN starts/restarts within their TA_GRACE period. This state is ACTive equivalent for the purpose of determining permissions.
RESTarting	T_SERVER object defined and currently being restarted by the system after an abnormal death. This state is ACTive equivalent for the purpose of determining permissions.
SUSpended	T_SERVER object defined and currently suspended pending shutdown. This state is ACTive equivalent for the purpose of determining permissions.
PARTitioned	T_SERVER object defined and active; however, the machine where the server is running is currently partitioned from the T_DOMAIN:TA_MASTER site. This state is ACTive equivalent for the purpose of determining permissions.

DEAd	T_SERVER object defined, identified as active in the bulletin board, but currently not running due to an abnormal death. This state will exist only until the BBL local to the server notices the death and takes action (REStarting|CLEaning). Note that this state will only be returned if the MIB_LOCAL TA_FLAGS value is specified and the machine where the server was running is reachable. This state is ACTive equivalent for the purpose of determining permissions.

SET: {NEW|INValid|ACTive|INActive|DEAd}

A SET operation will update configuration and run-time information for the selected T_SERVER object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_SERVER object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state.
unset	Modify an existing T_SERVER object. This combination is allowed only when in the ACTive or INActive state. Successful return leaves the object state unchanged.
INValid	Delete T_SERVER object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.
ACTive	Activate the T_SERVER object. State change allowed only when in the INActive state. (Servers in the MIGrating state must be restarted by setting the T_GROUP:TA_STATE to ACTive.) For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the ACTive state. The TMIB_NOTIFY TA_FLAG value should be used when activating a server if status on the individual server is required.

INACtive	Deactivate the T_SERVER object. State change allowed only when in the ACTive state. Successful return leaves the object in the INACtive state. The TMIB_NOTIFY TA_FLAG value should be used when deactivating a server if status on the individual server is required.
DEAd	Deactivate the T_SERVER object by sending the server a SIGTERM signal followed by a SIGKILL signal if the server is still running after the appropriate timeout interval (see TA_MIBTIMEOUT in MIB(5)). Note that by default, a SIGTERM signal will cause the server to initiate orderly shutdown and the server will become inactive even if it is restartable. If a server is processing a long running service or has chosen to disable the SIGTERM signal, then SIGKILL may be used and will be treated by the system as an abnormal termination. State change allowed only when in the ACTive or SUSpended state. Successful return leaves the object in the INACtive, CLEaning or RESTarting state.

TA_BASESRVID: 1 <= *num* < 30,001

Base server identifier. For servers with a TA_MAX attribute value of 1, this attribute will always be the same as TA_SRVID. However, for servers with a TA_MAX value greater than 1, this attribute indicates the base server identifier for the set of servers configured identically.

TA_CLASSPATH *string[0...2,047]*

The current CLASSPATH for the run time.

TA_CLOPT: *string[0...256]*

Command-line options to be passed to server when it is activated. See the servopts(5) reference page for details. Limitation: Run-time modifications to this attribute will not affect a running server.

TA_ENVFILE: *string[0...78]*

Server specific environment file. See T_MACHINE:TA_ENVFILE for a complete discussion of how this file is used to modify the environment. Limitation: Run-time modifications to this attribute will not affect a running server.

TA_GRACE: 0 <= *num*

The period of time, in seconds, over which the T_SERVER:TA_MAXGEN limit applies. This attribute is meaningful only for restartable servers, that is, if the

T_SERVER:TA_RESTART attribute is set to "Y". When a restarting server would exceed the TA_MAXGEN limit but the TA_GRACE period has expired, the system resets the current generation (T_SERVER:TA_GENERATION) to 1 and resets the initial boot time (T_SERVER:TA_TIMESTART) to the current time. A value of 0 for this attribute indicates that a server should always be restarted.

Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_JAVAHEAP 0<*num*
: The heap size as specified in the run-time options.

TA_JAVAHEAPUSE 1<=*num*<=100
: The percentage of heap space used.

TA_MAXGEN: 1 <= *num* < 256
: Number of generations allowed for a restartable server (T_SERVER:TA_RESTART == "Y") over the specified grace period (T_SERVER:TA_GRACE). The initial activation of the server counts as one generation and each restart also counts as one. Processing after the maximum generations is exceeded is discussed above with respect to TA_GRACE.

  Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

  Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_MAXDISPATCHTHREADS 0<*num*

The maximum number of threads, as specified with -M in the CLOPT.

TA_MAX: 1 <= *num* < 1,001

Maximum number of occurrences of the server to be booted. Initially, tmboot(3c) boots T_SERVER:TA_MIN objects of the server, and additional objects may be started individually (by starting a particular server ID) or through automatic spawning (conversational servers only). Run-time modifications to this attribute will affect all running servers in the set of identically configured servers (see TA_BASESRVID above) as well as the configuration definition of the server.

TA_MIN: 1 <= *num* < 1,001

Minimum number of occurrences of the server to be booted by. If a T_SERVER:TA_RQADDR is specified and TA_MIN is greater than 1, then the servers will form an MSSQ set. The server identifiers for the servers will be T_SERVER:TA_SRVID up to TA_SRVID + T_SERVER:TA_MAX - 1. All occurrences of the server will have the same sequence number, as well as any other server parameters.

Limitation: Run-time modifications to this attribute will not affect a running server.

TA_RCMD: *string[0...78]*

Application specified command to be executed in parallel with the system restart of an application server. This command must be an executable UNIX system file.

Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_RESTART: {Y|N}
: Restartable ("Y") or non-restartable ("N") server. If server migration is specified for this server group (T_DOMAIN:TA_OPTIONS/MIGRATE and T_GROUP:TA_LMID with alternate site), then this attribute must be set to "Y". Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

  Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_SEQUENCE: 1 <= *num* < 10,000
: Specifies when this server should be booted (tmboot(1)) or shutdown (tmshutdown(1)) relative to other servers. T_SERVER objects added without a TA_SEQUENCE attribute specified or with an invalid value will have one generated for them that is 10,000 or more and is higher than any other automatically selected default. Servers are booted by tmboot(1) in increasing order of sequence number and shutdown by tmshutdown(1) in decreasing order. Run-time modifications to this attribute affect only tmboot(1) and tmshutdown(1) and will affect the order in which running servers may be shutdown by a subsequent invocation of tmshutdown(1).

TA_SYSTEM_ACCESS: {FASTPATH|PROTECTED}
: Mode used by BEA Tuxedo system libraries within this server process to gain access to BEA Tuxedo system's internal tables. See T_DOMAIN:TA_SYSTEM_ACCESS for a complete discussion of this attribute.

  Limitation: Run-time modifications to this attribute will not affect a running server.

TA_CONV: {Y|N}
: Conversational server ("Y") or request/response server ("N").

TA_HWDISPATCHTHREADS
: The high water mark for the number of threads in the server.

TA_REPLYQ: {Y|N}

Allocate a separate reply queue for the server (TA_REPLYQ == "Y"). MSSQ servers that expect to receive replies should set this attribute to "Y".

TA_RPPERM: 0001 <= *num* <= 0777

UNIX system permissions for the server's reply queue. If a separate reply queue is not allocated (T_SERVER:TA_REPLYQ == "N"), then this attribute is ignored. Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_RQADDR: *string[0...30]*

Symbolic address of the request queue for the server. Specifying the same TA_RQADDR attribute value for more than one server is the way multiple server, single queue (MSSQ) sets are defined. Servers with the same TA_RQADDR attribute value must be in the same server group.

TA_RQPERM: 0001 <= *num* <= 0777

UNIX system permissions for the server's request queue.

Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_LMID: *LMID*

Current logical machine on which the server is running.

TA_GENERATION: 1 <= *num* < 32K

Generation of the server. When a server is initially booted via tmboot(1) or activated through the TM_MIB(5), its generation is set to 1. Each time the server dies abnormally and is restarted, its generation is incremented. Note that when T_SERVER:TA_MAXGEN is exceeded and T_SERVER:TA_GRACE has expired, the server will be restarted with the generation reset to 1.

TA_PID: 1 <= *num*

UNIX system process identifier for the server. Note that this may not be a unique attribute since servers may be located on different machines allowing for duplication of process identifiers.

Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

`TA_RPID`: 1 <= *num*
: UNIX system message queue identifier for the server's reply queue. If a separate reply queue is not allocated (`T_SERVER:TA_REPLYQ == "N"`), then this attribute value will be the same as `T_SERVER:TA_RQID`.

  Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

`TA_RQID`: 1 <= *num*
: UNIX system message queue identifier for the server's request queue. If a separate reply queue is not allocated (`T_SERVER:TA_REPLYQ == "N"`), then this attribute value will be the same as `T_SERVER:TA_RPID`.

  Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

`TA_THREADSTACKSIZE 0<`*num*
: The stack size per thread as specified in the runtime options.

`TA_TIMERESTART`: 1 <= *num*
: Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on `T_SERVER:TA_LMID`, when the server was last started or restarted.

`TA_TIMESTART`: 1 <= *num*
: Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on `T_SERVER:TA_LMID`, when the server was first started. Restarts of the server do not reset this value; however, if `T_SERVER:TA_MAXGEN` is exceeded and `T_SERVER:TA_GRACE` is expired, this attribute will be reset to the time of the restart.

`TA_NUMCONV`: 0 <= *num*
: Number of conversations initiated by this server via `tpconnect`(3c).

`TA_NUMDEQUEUE`: 0 <= *num*
: Number of dequeue operations initiated by this server via `tpdequeue`(3c).

`TA_NUMENQUEUE`: 0 <= *num*
: Number of enqueue operations initiated by this server via `tpenqueue`(3c).

`TA_NUMPOST`: 0 <= *num*
: Number of postings initiated by this server via `tppost`(3c).

`TA_NUMREQ`: 0 <= *num*
: Number of requests made by this server via `tpcall`(3c) or `tpacall`(3c).

TA_NUMSUBSCRIBE: 0 <= *num*

Number of subscriptions made by this server via tpsubscribe(3c).

TA_NUMTRAN: 0 <= *num*

Number of transactions begun by this server since its last (re)start.

TA_NUMTRANABT: 0 <= *num*

Number of transactions aborted by this server since its last (re)start.

TA_NUMTRANCMT: 0 <= *num*

Number of transactions committed by this server since its last (re)start.

TA_TOTREQC: 0 <= *num*

Total number of requests completed by this server. For conversational servers (T_SERVER:TA_CONV == "Y"), this attribute value indicates the number of completed incoming conversations. This is a run-time attribute that is kept across server restart but is lost at server shutdown.

TA_TOTWORKL: 0 <= *num*

Total workload completed by this server. For conversational servers (T_SERVER:TA_CONV == "Y"), this attribute value indicates the workload of completed incoming conversations. This is a run-time attribute that is kept across server restart but is lost at server shutdown.

TA_CLTLMID: *LMID*

Logical machine for the initiating client or server. The initiating client or server is the process that made the service request on which the server is currently working. The value in this field has meaning only for single-context servers. In multi-context servers, the null string is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_CLTPID: 1 <= *num*

UNIX system process identifier for the initiating client or server. The value in this field has meaning only for single-context servers. In multi-context servers, -1 is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

Limitation: This is a UNIX system-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_CLTREPLY: {Y|N}
: The initiating client or server is expecting a reply ("Y") or is not expecting a reply ("N"). The value in this field has meaning only for single-context servers. In multi-context servers, the null string is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_CMTRET: {COMPLETE|LOGGED}
: Setting of the TP_COMMIT_CONTROL characteristic for this server. See the description of the ATMI function call tpscmt(3c) for details on this characteristic. The value in this field has meaning only for single-context servers. In multi-context servers, the null string is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_CURCONV: 0 <= *num*
: Number of conversations initiated by this server via tpconnect(3c) that are still active. For multicontext servers, this field represents the total for all server contexts. Values for individual server contexts can be found in the T_SERVERCTXT class.

TA_CURDISPATCHTHREADS
: The current number of threads in the server

TA_CUROBJECTS
: The number of entries in use in the Bulletin Board object table for this server. Scope is local.

TA_CURINTERFACE
: The interface name of the interface currently active in this server. Scope is local.

TA_CURREQ: 0 <= *num*
: Number of requests initiated by this server via tpcall(3c) or tpacall(3c) that are still active. For multicontext servers, this field represents the total for all server contexts. Values for individual server contexts can be found in the T_SERVERCTXT class.

TA_CURRSERVICE: *string*[0 . . . 15]
: Service name that the server is currently working on, if any. The value in this field has meaning only for single-context servers. In multicontext servers, the null string is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_CURTIME: 1 <= *num*

Current time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_SERVER:TA_LMID. This attribute can be used to compute elapsed time from the T_SERVER:TA_TIMESTART and T_SERVER:TA_TIMERESTART attribute values.

TA_LASTGRP: 1 <= *num* < 30,000

Server group number (T_GROUP:TA_GRPNO) of the last service request made or conversation initiated from this server outward. The value in this field has meaning only for single-context servers. In multicontext servers, -1 is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_SVCTIMEOUT: 0 <= *num*

Time left, in seconds, for this server to process the current service request, if any. A value of 0 for an active service indicates that no timeout processing is being done. See T_SERVICE:TA_SVCTIMEOUT for more information. The value in this field has meaning only for single-context servers. In multicontext servers, -1 is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_TIMELEFT: 0 <= *num*

Time left, in seconds, for this server to receive the reply for which it is currently waiting before it will timeout. This timeout may be a transactional timeout or a blocking timeout. The value in this field has meaning only for single-context servers. In multicontext servers, -1 is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

TA_TRANLEV: 0 <= *num*

Current transaction level for this server. 0 indicates that the server is not currently involved in a transaction. The value in this field has meaning only for single-context servers. In multicontext servers, -1 is returned as a placeholder. This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

Limitations

None.