

CA Gen

Tuxedo User Guide

Release 8.5



This Documentation, which includes embedded help systems and electronically distributed materials (hereinafter referred to as the "Documentation"), is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2015 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen
- AllFusion® Gen
- COOL: Gen

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Contents

Chapter 1: Working With Tuxedo 9

Prerequisites	9
Related Documentation	9
Tuxedo Support with CA Gen	10
Tuxedo Distributed Processing Server	10
Tuxedo Distributed Processing Clients	11
Tuxedo DP Application Development and Deployment	11

Chapter 2: Tuxedo as a Transaction Processing Monitor 13

Applications with Transaction Control: XA-based Tuxedo Servers	13
Transaction	13
Distributed Transaction	13
CA Gen, Tuxedo, and Distributed Transaction Processing	14
Transaction Flows	15
Two Phase Commit	16
Applications with Transaction Control: Non-XA Tuxedo Servers	18
Failure Handling	19
Retry	19
Data Dependent Routing	20

Chapter 3: Detailing Server Managers for Tuxedo 21

Packaging Issues While Using Tuxedo	21
Invocation	22
Detailing the Server Managers for Tuxedo	23
Setting the Business System Server Environment	24
Setting the Individual Server Manager Properties	25

Chapter 4: Deploying the Tuxedo Application Servers 27

Tuxedo Administrator Tasks	27
Create a Transaction Management Server (TMS)	27
Setting Up the Server Environment	28
Configure the ENVFILE on the Server Machine	30
Configuring the UBBCONFIG File	30
Compiling the UBBCONFIG File	33
Creating the TLOG	34

Booting Up the Tuxedo Application Servers	34
Chapter 5: GUI, COM, and C Proxy Clients Using Tuxedo as the Communication Type	35
Adding WSL	35
Client Generation Considerations	36
Client Setup While Using Tuxedo	36
Building C Proxy Native Clients	38
Building C Proxy Workstation Clients.....	38
Chapter 6: Java-based Clients (Java Proxy or Web Generation) Using Tuxedo as the Communication Type	39
Server-Side Changes.....	39
Installing Tuxedo and Jolt.....	40
Adding JSL and JREPSVR	40
Enabling Tuxedo Servers for Java Client Access	40
Bulk Loading the Jolt Repository	41
Client-Side Changes.....	42
Modifying the commcfg.properties File.....	42
Environment Variables.....	42
Chapter 7: Tuxedo Proxy Client (ieftuxcl)	43
Startup Parameters	44
Environment Variables	44
Stopping ieftuxcl.....	45
Chapter 8: Block Mode Client (aeft)	47
Invoking the aeft	47
Environment Variables	48
AEFT Command Example	49
Chapter 9: Testing Tuxedo Applications	51
Using Diagram Trace Utility to Test Tuxedo Applications	51
Regeneration for Changes.....	52
Application Testing.....	53
Enabling Diagram Trace Utility.....	53
Accessing and Using Diagram Trace Utility	53
Multi-user Diagram Trace Utility Support (for Tuxedo servers communicating with IEFTUXCL only)	54

Regenerating Remote Files After Testing	55
-----------------------------------------------	----

Chapter 10: Components of a Typical CA Gen Tuxedo System **57**

Client-Side Components of a Typical CA Gen Tuxedo System.....	57
Windows Client	58
C Tuxedo Coopflow	58
Java Coopflow	59
Server-Side Components of a Typical CA Gen Tuxedo System.....	60
Server Runtime	61
Tuxedo Server Runtime	61
Server Manager	62
Components Involved in Server-to-Server Flow.....	62

Chapter 11: User Exit Functions **65**

C/C++ Client Runtime Exits.....	65
Server Runtime Exits	67
Tuxedo Server Runtime Exits	67

Chapter 12: Error Messages **69**

Client Error Messages	69
-----------------------------	----

Chapter 13: Flow Diagrams **71**

Chapter 14: Using the Application.ini File in Tuxedo **73**

Application.ini Contents	73
--------------------------------	----

Index **75**

Chapter 1: Working With Tuxedo

OracleTuxedo (referred to as Tuxedo) provides an execution environment for CA Gen client/server and block mode applications.

This guide includes design considerations for developing applications for Tuxedo, as well as information on generating, packaging, and deploying these applications. It also contains an overview on customizing user exits and guidelines for coding custom clients for use with Tuxedo.

The CA Gen download folder contains a number of sample files that are specific to Tuxedo; the location of these files is specified when a sample is referenced in the text.

This section provides a description of the Tuxedo features available with CA Gen and a general overview of the steps involved in designing, developing, and deploying CA Gen applications in a Tuxedo environment. Subsequent articles provide details of these steps.

This article is written for two audiences. Primarily, it is written for CA Gen application developers who are responsible for designing and developing client/server or block mode software applications, which will be deployed on a Tuxedo network. Additionally, it is written for system administrators who interact with the CA Gen developers.

Prerequisites

Developers should have a good working knowledge of CA Gen and some knowledge of Tuxedo.

System administrators should have an operational Tuxedo network in place at their site and a good working knowledge of their Tuxedo system.

A valid Tuxedo environment must be installed prior to generating your Tuxedo applications. If you are accessing Tuxedo Servers using Java-based clients, Oracle Jolt must be installed on the client and server systems.

Related Documentation

For additional help, see the CA Gen product documentation.

Note: For more information about the CA Gen product documentation, see the *Release Notes*.

The following outside sources may also provide useful background information:

- Tuxedo Reference documentation from Oracle.
- The Tuxedo System from Addison Wesley, ISBN 0-201-63493-7.

Tuxedo Support with CA Gen

CA Gen supports two types of Tuxedo application servers. Tuxedo application servers can either be CA Gen Block Mode applications or be CA Gen Distributed Processing Server (DPS) applications.

A CA Gen Block Mode application is a terminal based application that presents information in formatted screens. The formatted screens are presented to a terminal using CA Gen provided application known as the AEF Tuxedo Client (aef). The Block Mode Client (aef) chapter of this guide provides detailed information.

Note: For an introduction to the CA Gen Distributed Processing client/server application, see the *Distributed Processing - Overview Guide*.

Tuxedo provides a target execution environment for a CA Gen Distributed Processing Server (DPS) application.

The following sections provide information specific to Tuxedo Distributed Processing applications.

Tuxedo Distributed Processing Server

Tuxedo Application Servers generated with CA Gen use Tuxedo as a Transaction Processing (TP) Monitor. It is possible to deploy the Tuxedo Application Servers with or without transaction control.

- Application Servers that require transaction control can achieve it by utilizing the XA service interface from Tuxedo. This allows Tuxedo to perform Distributed Transaction management. Resource Management over the course of a distributed transaction is handled using the Two-Phase Commit process.
- If there are Application Servers where you do not want to use Transaction management, resource management is handled by CA Gen.

Tuxedo Distributed Processing Clients

The Tuxedo Application Servers can be accessed by a variety of clients. These include:

- Clients that provide a uniform interface for application communication, transaction support (for example, XA), and management of Tuxedo data buffers using the Tuxedo Application to Transaction Monitor Interface (ATMI). These may be traditional CA Gen GUI clients, Java Web Clients, or user-written clients using CA Gen proxy (C, COM, or Java).

For a CA Gen distributed application to exploit features provided by Tuxedo, the data that is exchanged is represented using the View32 data format of Tuxedo.

By employing the View32 data representation, CA Gen applications can use the data dependent routing feature of Tuxedo. To accomplish data dependent routing, the Tuxedo TP Monitor environment is configured to route data to different servers based on the content of data contained in the View32 buffer. Data dependent routing is achieved without modifying the CA Gen generated application code by using the capability provided by Tuxedo.

Additionally, Tuxedo Security is enabled through information provided by the user in the Tuxedo Configuration file (UBBCONFIG) and through making appropriate changes to a sample CA Gen runtime security user exit.

- Clients that use a Gen communications type of Gen or TCP/IP. Either of these clients needs to make use of the CA Gen Tuxedo Proxy Client (ieftuxcl). The Tuxedo Proxy Client provides a TCP/IP interface to the Tuxedo execution environment.

CA Gen clients that specify Gen as the communications type use the CA Gen Client Manager. The Client Manager can communicate with ieftuxcl directly or indirectly by using a CA Gen Communications Bridge.

Tuxedo DP Application Development and Deployment

The following are the steps necessary to develop and deploy a CA Gen Distributed Processing application. See the appropriate chapter for a more detailed explanation of each step.

1. Determine if clients will communicate with Tuxedo servers using the Tuxedo ATMI interface or the CA Gen Tuxedo Proxy Client. Using the Tuxedo Proxy Client implies the Gen DPC applications will use TCP/IP to communicate to the Tuxedo execution environment.
2. Design and develop the Tuxedo DP application.
3. If using Tuxedo ATMI to communicate between the DPC and DPS, a generated Tuxedo DPS application is capable of supporting server-to-server flows. Transactional behavior for server-to-server flows needs to be considered.

4. Specify Tuxedo properties using either Packaging or Generation. This includes packaging considerations and detailing Server Manager and individual PStep properties.
5. Generate and build the files for either a local or remote installation.
 - Remote and local generation is performed using the CA Gen Toolset. Remote generation is also performed using the CSE Construction Client.
 - You can build the application executable programs either locally or remotely.
6. Customize the client and server user exits as necessary.
7. Configure and deploy the Tuxedo Application Servers.
8. Configure the clients to access the Tuxedo Application Servers.
9. Test and run the cooperative application.

Chapter 2: Tuxedo as a Transaction Processing Monitor

Tuxedo Application Servers generated with CA Gen use Tuxedo as a Transaction Processing (TP) Monitor. It is possible to deploy the Tuxedo Application Servers with or without transaction control. Applications choosing Transaction Control do so through the XA interface of Tuxedo. Applications not choosing Transaction Control access their Resource Manager through CA Gen.

Applications with Transaction Control: XA-based Tuxedo Servers

This section describes applications that choose Transaction Control through the XA interface of Tuxedo.

Transaction

A transaction is defined as representing a set of operations (that is, a unit of work) that transform data from one consistent state to another, such that the change to the data has the following characteristics:

- **Atomic**-Either all operations that are part of the transaction happen or none happen.
- **Consistent**-Data modified by the operations of a transaction is transitioned to a consistent state.
- **Isolated**-Even though transactions may execute concurrently, no transaction has visibility to the work in progress of another transaction.
- **Durable**-After a transaction completes successfully, its change survives subsequent failures.

Distributed Transaction

A Distributed Transaction is a transaction that involves operations run in multiple processes, spread across one or more machines. Each process participates in the Distributed Transaction.

CA Gen, Tuxedo, and Distributed Transaction Processing

CA Gen Tuxedo support enables the use of Transaction Management in Tuxedo when an application is operating in a Server Procedure Step (PStep). With CA Gen, the Tuxedo Server runtime supports a Server PStep flowing to other Server PSteps. This support is accomplished through a PStep Use statement that establishes server-to-server flows. It is these server-to-server flows that operate and are controlled under the Transactional Management of Tuxedo.

A Tuxedo Server Manager PStep can either be the initial PStep in a transaction or a participant in the current transaction. If a current transaction does not exist when a PStep begins, the PStep creates a new (current) transaction. If a current transaction is in progress when the PStep begins, the PStep becomes a participant in the current transaction and so extends the transaction.

Invoking a Pstep is controlled through a model property. This property identifies if the PStep will participate in an existing distributed transaction and is defined using the Distributed Transaction Participant check box on the CA Gen Toolset or CSE PStep Properties dialog, shown in the following illustration.

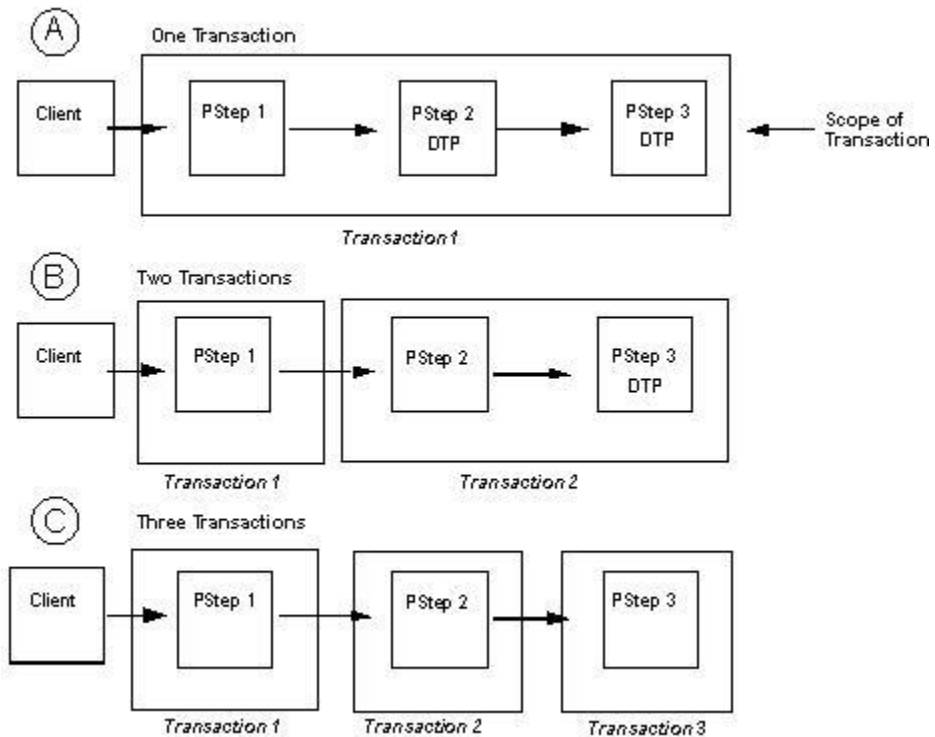
The screenshot shows the 'SERVER_MAINTAIN_EMPLOYEE Properties' dialog box with the 'TUXEDO' tab selected. The 'General' sub-tab is active. It contains several configuration fields: 'Names' with 'Source' set to 'SERVERM2' and an empty 'Screen' field; 'Trancodes' with 'Dialog Flow' set to 'P900' and 'Clear Screen' set to '<<NONE>>'; 'Dynamically Link (z/OS)' with 'Proc Step' and 'Screen' both set to 'Default'; and a section 'For TUXEDO, EJB and .NET Servers Only' containing an unchecked checkbox for 'Distributed Transaction Participant'. At the bottom are buttons for 'OK', 'Screen Control...', 'Cancel', and 'Help'.

If this check box is not checked (the default setting), the PStep is invoked in a non-transaction mode. Flows to this server will always execute under the control of a new transaction.

If the check box is checked, the PStep is invoked in transaction mode (if one exists) and the PStep extends the current transaction. If a current transaction does not exist when the PStep is invoked, a new transaction is created.

Transaction Flows

The following illustration shows the effect of defining Server PSteps as Distributed Transaction Participants (DTP) for three different distributed flows.



Note: Since PStep1 is the first server PStep in each flow, it always results in the creation of a new transaction. PStep1 is always defined as transactional regardless of whether Distributed Transaction Participant is set or not. The originating client is always outside the scope of transactional awareness.

Flow A-Shows the condition where PStep2 and PStep3 are individually detailed as Distributed Transaction Participants. The Distributed Transaction Participant check box is on.

Flow B-Shows the condition where PStep2 was detailed as **not** a Distributed Transaction Participant (default behavior). PStep2 breaks the flow into two transactions and creates a new transaction with PStep3 that has Distributed Transaction Participant set.

Flow C-Shows the condition where PStep2 and PStep3 are **not** set as Distributed Transaction Participants. In this case, the flow is separated into three transactions.

The commit or rollback operation for a Distributed Transaction is coordinated across all participating Procedure Steps. It is deferred until all the Procedure Steps participating in the transaction have completed.

Two Phase Commit

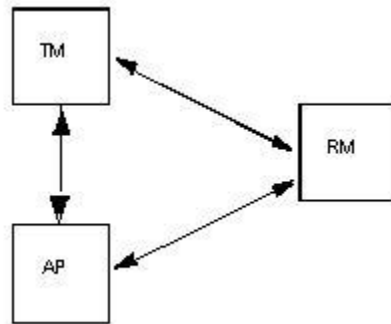
To understand how two phase commit is coordinated in a Tuxedo environment, you may find it useful to understand how the X/Open definitions for Distributed Transaction Processing relate to CA Gen.

X/Open Distributed Transaction Processing

The X/Open Distributed Transaction Processing (DTP) model defines the following separate entities that participate in a distributed application:

- Application Program (AP)
- Transaction Manager (TM)
- Resource Manager (RM)

The following illustration shows the relationships between these three entities:



The AP is the CA Gen server application code and it issues transactional statements to the RM. The TM is the Transaction Manager responsible for managing transactions on behalf of the application. The RM is the Resource Manager that typically is a database but can be any resource that needs management, such as a print spooler or a queue.

The interfaces between these components are defined in the X/Open DTP specification.

- The interface between the AP and RM is generally SQL, and no specification is given for this interface. At server startup, all application servers and Transaction Management Servers (TMSs) open a connection to the RM through the XA API `xa_open()`. This establishes a connection to the RM for global transactions. Conversely, at server shutdown the connection is closed by the XA API `xa_close()`. These XA APIs are managed through the Tuxedo APIs `tpopen()/tpclose()` which the AP invokes.

- The interface between the TM and RM is managed by Tuxedo. There are many procedural interfaces defined for XA compliant RMs (for example, `xa_open()`, `xa_start()`, `xa_end()`, `xa_prepare()`, `xa_commit()`, and `xa_recover()`). It is through these interfaces that the TM communicates with the RM, on behalf of the AP.
- The interface between the AP and TM is mainly through the Tuxedo transactional APIs `tpbegin()`, `tpcommit()` and `tpabort()`. One of the roles of the TM is to allocate a unique global transaction identifier, or XID. The XID is carried in every service call request made in this global transaction context. In this way, multiple services and servers can participate in the same global transaction. The various servers can belong to the same TMS group or to a different TMS group (for example, one server can belong to an Oracle TMS group while another server can belong to a TMS group). In each case, it is the responsibility of the RM to associate work for the same XID. When a request to commit is sent to the TMS the TMS begins the two phase commit process.

Two Phase Commit Process

The service that begins a global transaction is the service that commits or aborts the transaction. The TMS chosen to coordinate the commit is one from the server group that is first visited by the global transaction. As service calls proceed, a transaction participant tree is built of all servers involved in the transaction.

Phase 1

Prepare-The first phase of a two phase commit occurs when the coordinating TMS issues a prepare request. This is issued to all TMS servers, of all groups associated with the transaction participant tree for the global transaction. The request results in `xa_prepare(XID)` calls made by TMS servers to each associated RM. Phase 2 cannot begin until all RMs issue a successful acknowledgement. The results of the prepare requests are logged in the Tuxedo TLOG.

Phase 2

Commit-The second phase of a two phase commit occurs when the coordinating TMS issues a commit request to all participating TMS servers. This results in `xa_commit(XID)` calls made by TMS servers to their associated RMs. The results of the commit requests are logged in the Tuxedo TLOG.

If any failure acknowledgements are encountered during the commit phase, this constitutes a Heuristic Hazard, indicating that one or more RMs may be out of synchronization. When this happens, operator intervention is needed to explicitly commit or rollback the transaction.

Applications with Transaction Control: Non-XA Tuxedo Servers

There are certain cases where Tuxedo servers may not require the coordination of a Tuxedo Transaction Manager Server (TMS). For example, PSteps within a server only require read access to database tables. In this case there is no need to involve a TMS.

The XA activities, between a CA Gen Server Manager and the Tuxedo TM, can be circumvented by specifying a command line option associated with the UBBCONFIG entry of the server, and by placing these servers in a separate non-TMS server group.

The *GROUPS section of the UBBCONFIG file would look similar to:

```
*GROUPS
```

```
GROUP2 GRPNO=2
```

Adding the -oX option to the CLOPT parameter of a server causes the CA Gen Tuxedo Server Runtime to forgo all XA operations and will, instead, cause the Server application to connect and interact with the DBMS directly. The -o option informs the runtime that this server is non-XA and the X value associated with this option is ignored. Similarly, the command could be written as -oX, -oNONXA or -oanything.

The *SERVERS section of the UBBCONFIG file would look similar to:

```
*SERVERS
```

```
myserver SRVGRP=GROUP2 SRVID=1 CLOPT="-A -- -oX -t 15"
```

The Server Manager Runtime issues the database COMMIT and ROLLBACK statements, giving transactional control to the database. This interaction with the DBMS is handled in a similar way to how support for Tuxedo handled database interaction prior to COOL:Gen 5.0, that is, it makes use of the AEENV file. This AEENV file is read by the CA Gen runtime for database logon information.

The non-XA Tuxedo server cannot participate in an existing global transaction, as its database interaction is done without involving the Tuxedo TMS. Server-to-server flows within the same transaction are not possible. The Distributed Transaction Participant attribute associated with the PSteps within a non-XA server is ignored.

Failure Handling

In CA Gen, there are two distinct methods for handling the failure of a PStep. The failure can be treated either as an abend or a rollback. If the situation is treated as a rollback, the export view is returned and the PStep that fails does not force the calling PStep to abend. If the failure is treated as an abend, there are a minimum of four scenarios that may cause the condition to occur:

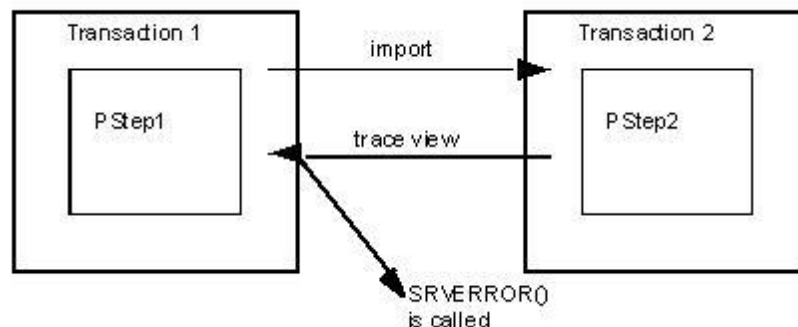
- An Application PStep sets an Exit State of ABORT
- Communications failure
- Unhandled database (DBMS) error
- The application issues an Abort Transaction action language statement

If an abend occurs, the trace view is returned to the calling PStep. The trace view contains an XFAL buffer message, so that, if PStep2 fails, the resulting XFAL message causes PStep1 to fail as well.

By using the following action statement, abend scenario 3 can be processed as a rollback. This change gives control to the application instead of the runtime handling the abend.

```
WHEN DBMS ERROR
EXIT STATE is ROLLBACK
ESCAPE
```

When an abend occurs, the invoking server is notified of the failure and control is given to `SRVERROR()` that is the server error exit. This exit allows viewing of the contents of the XFAL message, and to amend it if necessary.



Retry

Before transactional behavior was supported, the CA Gen Runtime automatically retried PSteps that failed due to database lockout conditions. Several mechanisms exist to modify the default retry behavior of the runtime.

In a transactional environment, it is important to avoid the possibility of retrying an action that was already committed, so you should ensure that `RETRY_LIMIT` is set to 0. `RETRY_LIMIT` may be set in any or all of the following places:

- In the server environmental variable `RETRY_LIMIT`. (This variable controls all unless you override it.)
- In the `tirdrtl.c` (default retry limit) exit located in the `$IEFH/src` directory. (If present, this overrides the environmental variable settings.)
- In the action language. (PStep level control)

Set `RETRY_LIMIT` to 0 for all transactional processing.

Data Dependent Routing

Both the developer and the system administrator have roles to play in implementing data dependent routing. The developer needs to be aware that the data dependent routing can be accomplished by setting ranges on a given view member.

The developer also needs to provide the system administrator with the Tuxedo View Name and View Member that is used to determine routing and the criteria (data ranges) needed to plan the data routes. This information is needed to configure the `UBBCONFIG` file to support data dependent routing.

Chapter 3: Detailing Server Managers for Tuxedo

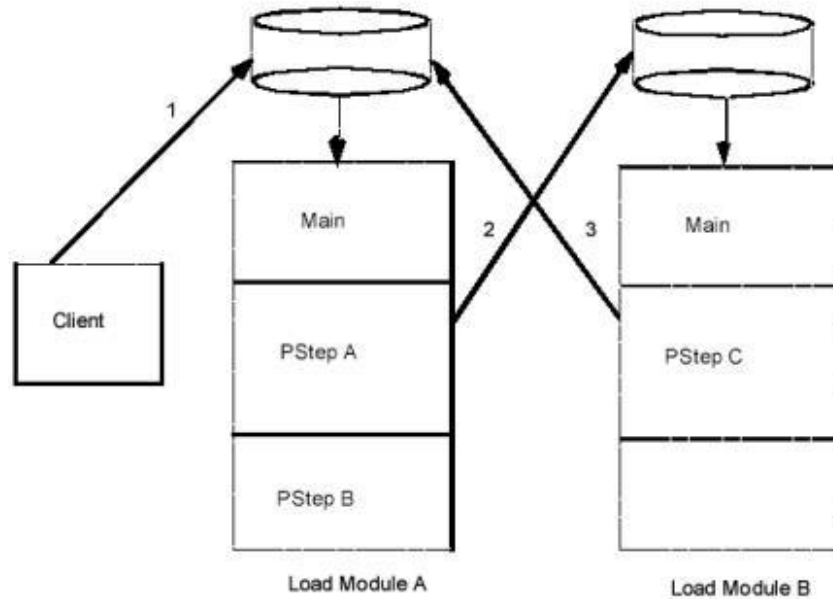
Packaging Issues While Using Tuxedo

A developer must pay special attention to packaging since certain Tuxedo features are affected by how an application is packaged. Some possible areas of consideration in packaging include:

- How you can most effectively use packaging to take advantage of transactional PSteps in the Tuxedo environment.
- PSteps that flow to one another and are packaged in the same load module must operate within the scope of the same transaction.
- You need to be aware that a deadlock situation can be created by the way that load modules are packaged. For example, if a called remote PStep calls back to another PStep in the original calling Load Module. See the section Invocation Deadlock for details of packaging conditions that can result in a Tuxedo Runtime deadlock.

Invocation

The following illustrates a situation where PStepA and PStepB are both packaged in the same Server Load Module. The result of this deadlock situation is a timeout for PStepA and PStepC.



The following flow results in the deadlock:

1. Client calls PStep A.
2. PStep A calls PStep C and waits for a response.
3. PStep C calls PStep B and waits for a response. However, PStep B cannot execute until PStep A completes.

To prevent this situation, you must either package PSteps so this cannot occur, or properly configure the UBBCONFIG file of the Tuxedo Multiple Servers Single Queue (MSSQ) feature.

Note: For more information about configuring the UCCCONFIG file for Tuxedo Multiple Servers Single Queue, see the *OracleTuxedo Reference Manual*.

Detailing the Server Managers for Tuxedo

To specify Server Managers to be built as Tuxedo Application Servers, it is essential to set TP Monitor to **Tuxedo** while detailing the Server Managers. The Communications can be set to either **Gen** or **Tuxedo** depending on the type of clients that will access the specific Server Manager. Server Managers are detailed through either the Toolset or the CSE Construction Client. Either method is complete and independent of the other.

You can globally specify Server Manager environment default values by selecting a Business System, then detailing the Server Environment. These default values apply to all Server Managers that do not have specifically detailed server environment values. You can also detail each Server Manager individually through the Server Environment Parameters dialog.

When using Server Managers detailed for Tuxedo, you have the option of setting properties for any or all of the PSteps associated with a certain Server Manager.

If you detail individual PStep properties, these properties are saved in the model. The procedure step properties specific to Tuxedo are specified using two tabs of the procedure step properties dialog:

- **General tab - Distributed Transaction Participant**

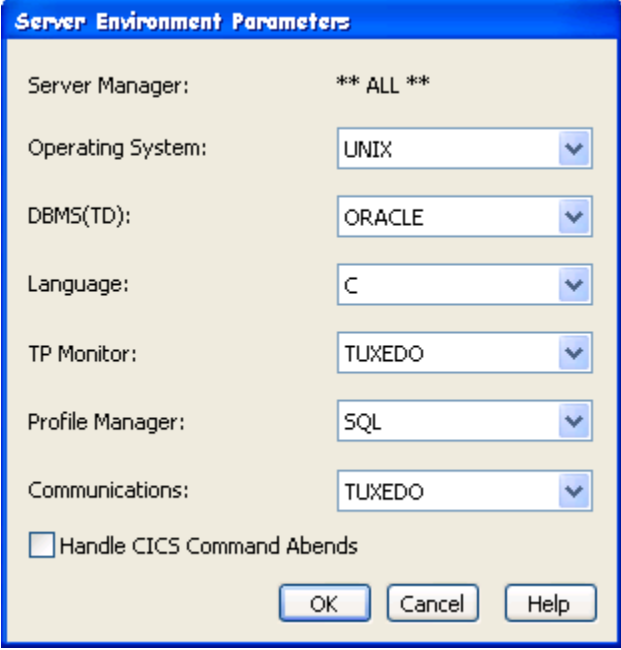
By default, the Distributed Transaction Participant check box is **not** checked. In this case, the associated PStep begins a new transaction.

- **Tuxedo tab - Tuxedo Service Name**

By default, this name is generated from the PStep source file name. You may overwrite this value if you want to use names conforming to the standards of your company, or if you anticipate any conflict of file name within the Tuxedo environment.

Setting the Business System Server Environment

The Server Environment is specified for all the Server Managers that constitute a Business System by highlighting a Business System, and then selecting Detail, Server Environment from the right-click popup menu inside the Cooperative Code Generation Window. The Server Environment Dialog that displays the Server Environment for all the Server Managers is shown next.



The screenshot shows a dialog box titled "Server Environment Parameters". It contains several configuration options, each with a label and a value field. The "Server Manager:" field is set to "** ALL **". The "Operating System:" field is set to "UNIX". The "DBMS(TD):" field is set to "ORACLE". The "Language:" field is set to "C". The "TP Monitor:" field is set to "TUXEDO". The "Profile Manager:" field is set to "SQL". The "Communications:" field is set to "TUXEDO". There is a checkbox labeled "Handle CICS Command Abends" which is currently unchecked. At the bottom right, there are three buttons: "OK", "Cancel", and "Help".

Parameter	Value
Server Manager:	** ALL **
Operating System:	UNIX
DBMS(TD):	ORACLE
Language:	C
TP Monitor:	TUXEDO
Profile Manager:	SQL
Communications:	TUXEDO

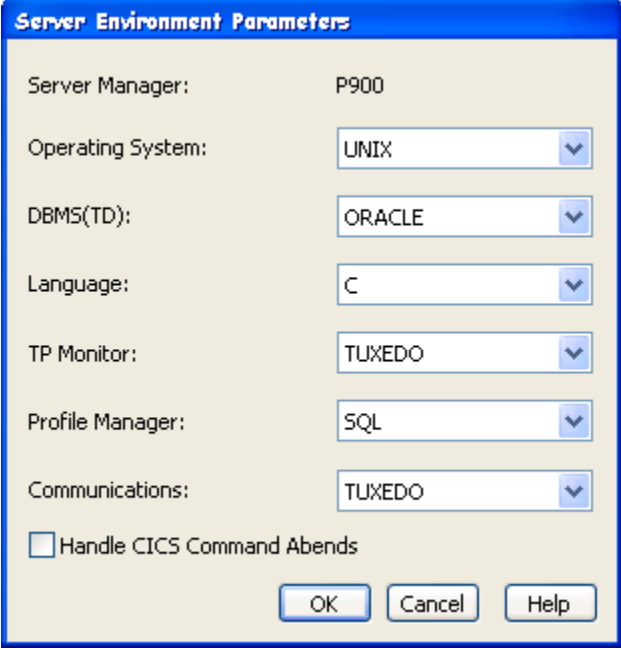
☐ Handle CICS Command Abends

OK Cancel Help

Note: The **ALL** indicates that the specified server environment parameters define default values that apply to all the server managers within the associated Business System that do not have explicitly detailed server environment parameters.

Setting the Individual Server Manager Properties

The Server Environment is specified for an individual Server Manager by highlighting a Server Manager, and then selecting Detail, Server Environment from the right click popup menu, inside the Cooperative Code Generation Window. The Server Environment Dialog that displays the Server Environment for a particular Server Manager is shown next.



Server Manager:	P900
Operating System:	UNIX
DBMS(TD):	ORACLE
Language:	C
TP Monitor:	TUXEDO
Profile Manager:	SQL
Communications:	TUXEDO
<input type="checkbox"/> Handle CICS Command Abends	
OK Cancel Help	

Chapter 4: Deploying the Tuxedo Application Servers

After building the server application, the output files from the build are located in the `inqload` subdirectory. These files are the *loadmodule* and the *PStep view* file, both of which are needed for the Tuxedo configuration.

Tuxedo Administrator Tasks

The Tuxedo administrator has to set up some of the environment so the Tuxedo Application Servers can be deployed. The following section details some of these tasks.

The Oracle Tuxedo subsystem uses semaphores, message queues, and shared memory Interprocess Communication resources provided by the UNIX operating systems. UNIX systems often benefit from customization of kernel parameters associated with these IPC resources.

The names and recommended values for IPC tuning parameters may vary depending on the operating system vendor and the specific release. See the Oracle documentation for information on configuring UNIX kernel parameters for the Oracle Tuxedo system.

It is important to consider the message queue parameters `MSGMAX` and `MSGMNB`. CA Gen Tuxedo applications support data transfers up to 32 KB in size. This 32 KB buffer size should be considered when adjusting these two kernel parameters.

Create a Transaction Management Server (TMS)

Tuxedo uses Transaction Manager Servers (TMS) for distributed transaction processing. You must create a TMS for each Resource Manager (RM). The RM must conform to the X/OPEN XA interface.

Note: This is only needed for Tuxedo Application Servers utilizing XA. Not all RMs are XA compliant. RMs that are XA compliant include Oracle.

Typically, the Tuxedo administrator builds the TMS for each database before any CA Gen applications are built (that is, compiled and linked) on the server machine.

Building a TMS involves editing the file `$TUXDIR/udataobj/RM`. This file contains the list of libraries required for linking with the appropriate RM. This file is also used during the Gen server build process, so if the TMS is not built successfully, then the server build fails as well. The environment variables `PATH`, `TUXDIR` and `SHLIB_PATH` should be set before building the TMS.

Note: `SHLIB_PATH` is the system dependent environment variable used to indicate the location of the shared libraries. Use `LD_LIBRARY_PATH` on Solaris, `LIBPATH` on AIX.

Sample RM File

```
# Copyright (C) 2009 CA, All rights reserved.
# RM file for HP-UX:
# Oracle 11g and DB2 9.5
# Copy this file into udataobj directory of your Tuxedo installation
# and rename it to RM, prior to running buildtms command
# ORACLE 11g
Oracle_XA:xaosw:-L${ORACLE_HOME}/lib -lclntsh -lsqll1 -lpthread
# DB2 9.5
Db2_XA:db2xa_switch:-L{DB2DIR}/lib -ldb2 -lhppa -lm -ldld
Oracle_XA: The RM for Oracle.
A TMS for Oracle_XA can be built as follows:
$TUXDIR/bin/buildtms -o TMS_Oracle101 -r Oracle_XA
TMS_Oracle101: The TMS load module that is created as a result of the above stwp.
You need to move this to the $TUXDIR/bin directory.
```

Setting Up the Server Environment

The following environment variables must be set on the server systems while bringing up the Tuxedo Application Servers.

Variable	Value
TUXDIR	Set to the directory where Tuxedo is installed. Example: setenv TUXDIR/opt/tuxedo/9.1
PATH	Set to the search path for commands. This should include <code>\$TUXDIR/bin</code> . Example: setenv PATH \$TUXDIR/bin:\$PATH
INCLUDE	Set to the directory for Tuxedo header files. This should include <code>\$TUXDIR/include</code> . Example: Setenv INCLUDE \$TUXDIR/include;\$INCLUDE

Variable	Value
SHLIB_PATH	<p>Set to the directory for Tuxedo library files. This should include \$TUXDIR/lib. Example: setenv SHLIB_PATH \$TUXDIR/lib;\$SHLIB_PATH</p> <p>Note: This is a system dependent environment variable used to indicate the location of the shared libraries. Use LD_LIBRARY_PATH on Solaris, LIBPATH on AIX.</p>
TUXCONFIG	<p>Set to the full pathname of the binary configuration file that results from the UBBCONFIG file described in the following sections. Example: setenv TUXCONFIG=<model directory>/inqload/tuxconfig</p>
VIEWDIR32	<p>Set to a semi-colon separated list of directories specifying the directory location of the view object files. Example: setenv VIEWDIR32=<model directory>/inqload</p> <p>Note: This is not needed for clients using TCP/IP or CA Gen as the communication type. See the chapter "Tuxedo Proxy Client (ieftuxcl)" for setup information.</p>
VIEWFILES32	<p>Set to a comma-separated list of view object filenames generated for the application. Example: VIEWFILES32=<Pstep1>.V, <Pstep2>.V,...</p> <p>Note: This is not needed for clients using TCP/IP or CA Gen as the communication type. For setup information, see the chapter "Tuxedo Proxy Client (ieftuxcl)."</p>
AEPATH	<p>Search path for executable load modules. Set to a semicolon-delimited list of directory paths that indicate a search order for the user's application directories. Example: AEPATH=[directory_1;directory_2;...directory_n]</p> <p>Note: It is expected that each directory on the AEPATH contains an inqload directory where the executable load modules should reside.</p>
CLASSPATH	<p>Set of colon separated jolt jar files (jolt.jar, joltadmin.jar, joltjse.jar). Example: CLASSPATH=\$TUXDIR/udataobj/jolt/jolt.jar:....:\$CLASSPATH</p> <p>Note: This is needed for enabling Java-based clients access to Tuxedo Application Servers, either through the Java Proxy or Web Generation.</p>

Some of these variables can also be set through the ENVFILE file as described in the following sections

Configure the ENVFILE on the Server Machine

All Tuxedo servers on the server machine are executed with the environment specified in the ENVFILE. This path or filename also is specified in the *MACHINES section of the UBBCONFIG file.

The following code sample is a self-documented sample file which you may use as a template to configure your site specific ENVFILE. See your Tuxedo documentation for more specifics on the ENVFILE. A similar template ENVFILE is included in the \$IEFH/make directory.

Sample ENVFILE File

```
#Copyright (C) 2009 CA, All rights reserved.
#sample ENVFILE. Rename this file following modification
#Replace the <bracketed> items with the appropriate values
#List all of the binary View32 files (comma separated list)
VIEWFILES32=<view32 file>.V,<view32 file>.V
VIEWDIR32=<Target directory>
#List all the Gen required SHLIB_PATH directory
#In particular, $IEFH/make directory
SHLIB_PATH=<List all the required SHLIB_PATH list>
#Retry should not be used in distributed transaction
RETRY_LIMIT=0;
#For debugging purpose
#CMIDEBUG=0xFFFFFFFF
```

Note:

- Replace <view32 file>.V with a comma- separated list of View file names.
- SHLIB_PATH is a system dependent environment variable used to indicate the location of the shared libraries. Use LD_LIBRARY_PATH on Solaris, LIBPATH on AIX.

Configuring the UBBCONFIG File

You need to configure several sections of the UBBCONFIG file to allow CA Gen generated clients and servers to interact in a Tuxedo environment. See the OracleTuxedo documentation on minimum Tuxedo specific requirements, and the CA Gen specific requirements described in the UBBCONFIG template file.

A commented UBBCONFIG template is located in \$IEFH/make directory. The following sample UBBCONFIG file illustrates a one server implementation. Pay special attention to the SERVERS and SERVICES sections.

Sample UBBCONFIG File

```

# Copyright (C) 2009 CA, All rights reserved.
# Sample ubbconfig file for Gen application
# Replace the <bracketed> items with the appropriate values.
# Rename this file following any modification
*RESOURCES
IPCKEY                <Replace with a valid IPC Key>

#Example:
#IPCKEY                123456
DOMAINID              GENAPP
MASTER                ADVANTAGE
MAXACCESSERS          15
MAXSERVERS             10
MAXSERVICES            20
MODEL                  SHM
LDBAL                  Y
#Change security level as necessary
SECURITY               NONE
*MACHINES
DEFAULT:
ENVFILE="<Replace with your ENVFILE pathname>"
APPDIR="<Replace with the application directory pathname>"
TUXCONFIG="<Replace with your TUXCONFIG pathname>"
TUXDIR="<Replace with your Tuxedo installation directory>"
MAXWSCLIENTS=<replace with the MAX NUMBER OF WS CLIENTS>
TYPE="hpux"
TLOGDEVICE="<Replace with your TLOG pathname>"
TLOGNAME="TLOG"
TLOGSIZE=10

#Example:
#   ENVFILE="/users/tony/GENAPP/inqload/ENVFILE"
#   APPDIR="/users/tony/GENAPP/inqload"
#   TUXCONFIG="/users/tony/GENAPP/inqload/tuxconfig"
#   TUXDIR="/opt/tuxedo"
#   MAXWSCLIENTS=9
#   TYPE="hpux"
#   TLOGDEVICE="/users/tony/IT/TEST/inqload/TLOG"
#   TLOGNAME="TLOG"
#   TLOGSIZE=10
<Machine-name>        LMID=ADVANTAGE
#Example:
#salesunx              LMID=ADVANTAGE
*GROUPS
GROUP1
    LMID=ADVANTAGE    GRPNO=1 TMSNAME="TMS_Oracle" TMSCOUNT=2

```

```
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/<Oracle username>/<Oracle
password>+S
esTm=60"
GROUP2
    LMID=ADVANTAGE    GRPNO=2
#Example of OPENINFO
#for Oracle:
#OPENINFO="Oracle_XA:Oracle_XA+Acc=P/tony/tonypassword+SesTm=60"
#Create one entry in *SERVER section for every server load module
#you wish to deploy
*SERVERS
#XA Server.
<SERVER LOAD MODULE NAME> SRVGRP=GROUP1 SRVID=1
                        RQADDR=<Give a unique queue name for each
                        server LM>
                        REPLYQ=Y
                        CLOPT="-A -- -t 15"

#Non-XA Server
<SERVER LOAD MODULE NAME> SRVGRP=GROUP2 SRVID=1
                        RQADDR=<Give a unique queue name for each
                        server LM>
                        REPLYQ=Y
                        CLOPT="-A -- -0"

#Example:
#SERVER1                SRVGRP=GROUP1 SRVID=1
#                        RQADDR=SERVER1_Q
#                        REPLYQ=Y
#                        CLOPT="-A -- -t 15"
#
#SERVER2                SRVGRP=GROUP2 SRVID=1
#                        RQADDR=SERVER2_Q
#                        REPLYQ=Y
#                        CLOPT="-A -- -0"
#Add security server if needed
#AUTHSVR                SRVGRP=GROUP1 SRVID=23 RQADDR=AUTH_Q
REPLYQ=Y
#Workstation Listener Server
```



```
WSL          SRVGRP=GROUP1 SRVID=21 RQADDR=WSL_Q  REPLYQ=Y          CLOPT="-A
-- -n <Address and port no.>
#Example of WSL network address using TCP:
#CLOPT="-A -- -n 0x00029000AC193709"
#TCP: 0002          Port: 9000          IP address: AC 19 37 09 (172.25.55.9)
# Jolt Listener (JSL) Server.
# To be set for Tuxedo server's that need to be accessed using the Jolt API
JSL          SRVGRP=GROUP1 SRVID=21 RQADDR=JSL_Q  REPLYQ=Y
          CLOPT="-A -- -n //hostName:portNumber"
# Jolt Repository Server (JREPSVR)
# The server that handles the jolt repository which stores the Server Interfaces
# used by the Jolt based clients. RepositoryFileName specifies the repository file
JREPSVR SRVGRP=GROUP1 SRVID=22 RESTART=Y MAXGEN=3 GRACE=0
          CLOPT="-A -- -W -P repositoryFileName"
```

Compiling the UBBCONFIG File

The UBBCONFIG file that was created must be compiled into a binary version for use by the Tuxedo BB. This can be done as follows:

```
% tmloadcf -y UBBCONFIG
```

It is important that the environment variable TUXCONFIG is set to the same value as directed by the TUXCONFIG directive in the UBBCONFIG file.

Creating the TLOG

The next step is to create a TLOG file. This is done as follows:

```
%  
  
tadmin - Copyright (c) 1996-2010 Oracle.  
Portions * Copyright 1986-1997 RSA Data Security, Inc.  
All Rights Reserved.  
Distributed under license by Oracle.  
Tuxedo is a registered trademark.  
No bulletin board exists. Entering boot mode.  
  
> crdl -z <TLOGDEVICE> -b 200  
Device created: <TLOGDEVICE>  
  
> crlog -m <hostname>  
crlog successfully completed.  
  
> quit  
%
```

TLOGDEVICE is the full path name for the log file, and should be the same as the value as directed by the TLOGDEVICE directive in the UBBCONFIG file.

Booting Up the Tuxedo Application Servers

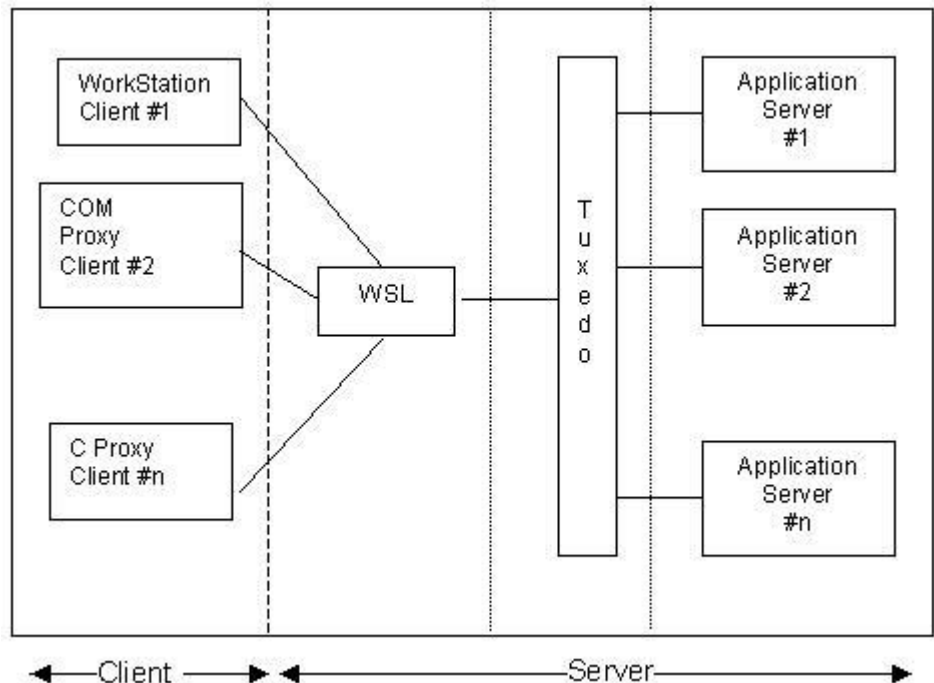
The Tuxedo BB and the Tuxedo Application Servers can be brought up as follows:

```
% tmbboot -y UBBCONFIG
```

Chapter 5: GUI, COM, and C Proxy Clients Using Tuxedo as the Communication Type

Tuxedo clients are referred to as either Native or Workstation clients. Native clients reside on the same system as a Tuxedo server and share the same configuration. Workstation clients reside on a remote machine that does not have either a bulletin board or a Tuxedo administration server. Workstation clients communicate using a Workstation Listener (WSL).

The following diagram illustrates the components involved in this kind of setup.



Adding WSL

The GROUPS and the SERVERS section in the UBBCONFIG file must be modified to add a new server called the Workstation Listener (WSL).

The following must be completed before a Client using Tuxedo as the communication type can access the Tuxedo servers.

Sample UBBCONFIG File

```
...
*GROUPS
...
WSL_GROUP GRPNO=1
...
*SERVERS
...
#Workstation Listener Server
WSL
    SRVGRP=WSL_GROUP
    SRVID=1
    REPLYQ=Y
    CLOPT="-A - - -n//wslHostName:wslPortNumber"
```

Client Generation Considerations

Before generating the Clients (GUI, COM, or C Proxy), ensure that you have set the Server Environment Parameters (TP Monitor and Communications) to TUXEDO. When set, generation includes the Tuxedo Application Server viewfiles. The client needs these viewfiles to communicate with the Tuxedo Application Servers.

Client Setup While Using Tuxedo

The following environment variables must be set on the client systems in order to access the Tuxedo Application Servers. While the clients can be either Native or Workstation, only the WSNADDR variable is applicable to Workstation clients. All other variables are for Native clients.

Variable	Value
TUXDIR	Set to the directory where Tuxedo is installed. Example: TUXDIR=c:\tuxedo
TUXCONFIG	Set to the full path name of the binary configuration file that results from the UBBCONFIG file. Example: TUXCONFIG=\$MODELDIR/inqload/tuxconfig
PATH	Set to the search path for commands. This should include %TUXDIR%/bin. Example: PATH=%TUXDIR%\bin;%PATH%

Variable	Value
INCLUDE	Set to the directory for Tuxedo header files. This should include %TUXDIR%\include Example: INCLUDE=%TUXDIR%\include;%INCLUDE%
LIB	Set to the directory for Tuxedo library files. This should include %TUXDIR%\lib. Example: LIB=%TUXDIR%\lib;%LIB%
WSNADDR	Set to indicate the Host name and the Port number where the Tuxedo WSL is running. Example: WSNADDR=//razor:7878, where razor is the machine name of the server and 7878 is the port number on which the WSL is listening.
VIEWDIR32	Set to a semi-colon separated list of directories specifying the directory location of the view object files. Example: VIEWDIR32=c:\model\sample
VIEWFILES32	Set to a comma- separated list of view object filenames generated for the application. Example: VIEWFILES32=<Pstep1>.VV, <Pstep2>.VV,...
AEPATH	Search path for executable load modules. Set to a semicolon-delimited list of directory paths that indicate a search order for the user's application directory(s). Example: AEPATH=[directory_1;directory_2;...directory_n] Note: It is expected that each directory on the AEPATH contains an inqload directory where the executable load modules should reside.

Building C Proxy Native Clients

The following set of commands can be used to build the native clients for C proxies.

Example

```
LIB=-L$(IEFH)/lib -lpxrt.XX  
CC=$(CC)  
CFLAGS="<whatever other flags you need> $(LIB)"  
Buildclient -o nativeClient -f nativeClient.c
```

Note: xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Building C Proxy Workstation Clients

The Gen Tuxedo Workstation Client library needs to be linked in so that it concatenates ahead of the Tuxedo Native Clients library.

Example

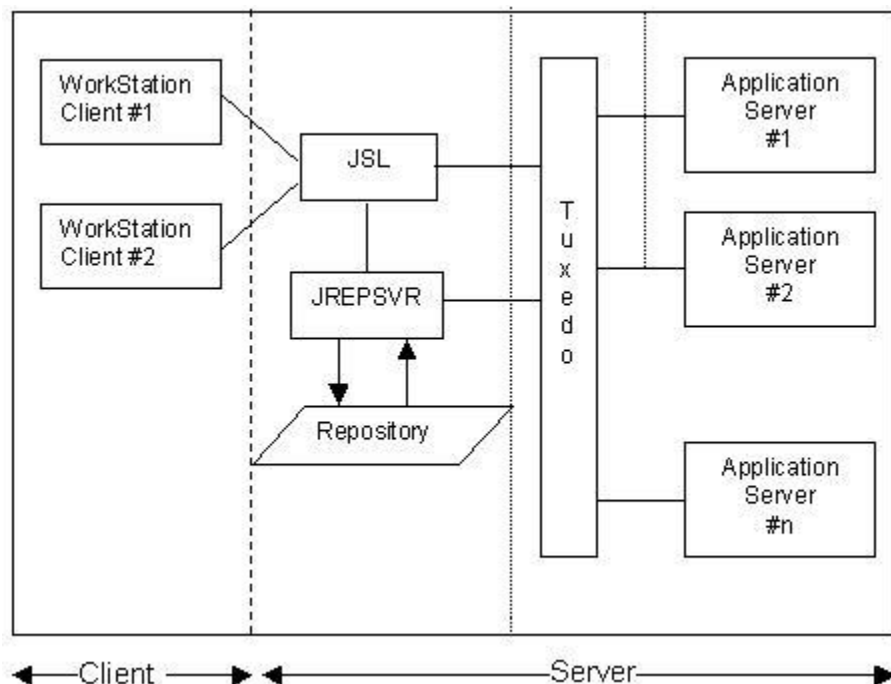
```
LIB=-L$(IEFH)/lib -lpxrt.XX -ltxwcf.XX  
CC=$(CC)  
CFLAGS="<whatever other flags you need> $(LIB)"  
Buildclient -w -o WSCClient -f WSCClient.c
```

Note: xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Chapter 6: Java-based Clients (Java Proxy or Web Generation) Using Tuxedo as the Communication Type

It is possible to target Tuxedo servers using Java-based clients (Java Proxy or Web Generation Clients). Using the Jolt product from Oracle enables this.

The following diagram illustrates the components involved in this kind of setup.



Server-Side Changes

The steps in the following sub sections must be completed before a Java-based client can access the Tuxedo servers.

Installing Tuxedo and Jolt

The System Administrator must install the Oracle Jolt product on the server systems that use Tuxedo servers and it must be accessible by Java-based clients. This is in addition to the Tuxedo product that must be installed to deploy Tuxedo servers.

Adding JSL and JREPSVR

The GROUPS and the SERVERS sections in the UBBCONFIG file must be modified to add two new servers called the Jolt Server Listener (JSL) and the Jolt Repository Server (JREPSVR).

Sample UBBCONFIG File

```
...
*GROUPS
#Jolt related Groups.
JSL_GROUP GRPNO=4
REPSVR_GROUP GRPNO=5
...
*SERVERS
...
#Jolt related Servers
JSL
  SRVGRP=JSL_GROUP
  SRVID=21
  RQADDR=JSL_Q
  REPLYQ=Y
  CLOPT="-A -- -n//jslHostName:jslPortNumber"

JREPSVR
  SRVGRP=REPSVR_GROUP
  SRVID=21
  RESTART=Y
  CLOPT="-A -- -W -P RepositoryFileName"
...
```

Enabling Tuxedo Servers for Java Client Access

You can build Tuxedo Servers for access by traditional C clients, with some minor changes as follows:

- The {PStep}.tvf file compile results are stored in a {PStep}.TV file.
- A symbolic link, {PStep}.V, is created to point to {PStep}.TV.

In addition, all the information that is needed for Java client access to the Tuxedo Servers is generated in the form of two files: {PStep}.sh and {PStep}.jvf. Users must explicitly enable the Tuxedo Servers for access by Java-based clients as follows:

- Shutdown the Tuxedo Servers, if they are up and running.
- Execute all the {PStep}.sh scripts for all of the affected servers. These scripts:
 - Compile the {PStep}.jvf to obtain the {PStep}.JV files.
 - Copy the {PStep}.JV files to the inqload directory.
 - Change the symbolic link {PStep}.V to point to {PStep}.JV.

Notes:

Only C clients can access Tuxedo Servers if the symbolic link {PStep}.V points to {PStep}.TV.

Both C and Java clients can access Tuxedo Servers if the symbolic link {PStep}.V points to {PStep}.JV.

Bulk Loading the Jolt Repository

When the server remotes (.RMT files) are split and built, it results in the generation of some Tuxedo specific files for each server load module. These files have the extensions .tvf, .jvf, and .sh.

The .jvf files make the Tuxedo servers accessible to Java-based clients. The .tvf and the corresponding .jvf files both describe the views that are used by the Tuxedo server, but differ in some of the field names and field types.

The .jblk files are the result of the compilation of the .jvf file, which takes place when the .sh files are executed by the user to enable access by Java-based clients to Tuxedo Servers. The .jvf can also be compiled into the .jblk as follows:

```
$IEFH/make/jvf2jblk.sh JvfFile.jvf
```

Before continuing, the Jolt servers (JSL and JREPSVR) must be started.

The data in the .jblk file is in a format that is understood by the Jolt Bulk Loader. The Tuxedo Administrator needs to load the information from the .jblk file into the Jolt Repository, using the Bulk Loader tool provided with the Jolt installation.

```
java oracle.jolt.admin.jbld -p package //jslHostName:jslPortNumber JblkFileName
```

Where package could be a Name under which the Services are grouped.

Client-Side Changes

For client side changes, the steps in the following sub sections must be completed.

Modifying the commcfg.properties File

JSL_NETWORK_ADDRESS is a variable in the file commcfg.properties. This must be changed to point to the host and the port on which the JSL is running. The commcfg.properties file is typically located in the InstallDir/Gen directory on the client system.

Sample commcfg.properties File

```
...
#####TUXEDO / JOLT#####
#
# The commcfg.properties file can be used to set the Jolt System Network Address
# (JSL_NETWORK_ADDRESS) to point to the Remote System & Port Number hosting
# the JSL.
#
# This value in turn can be overridden by modifying the Security Exit that is
# invoked prior to the flow.
#
#####
JSL_NETWORK_ADDRESS=//localhost:portNumber
...
where localhost can be in IPv4 or IPv6 format.
```

Environment Variables

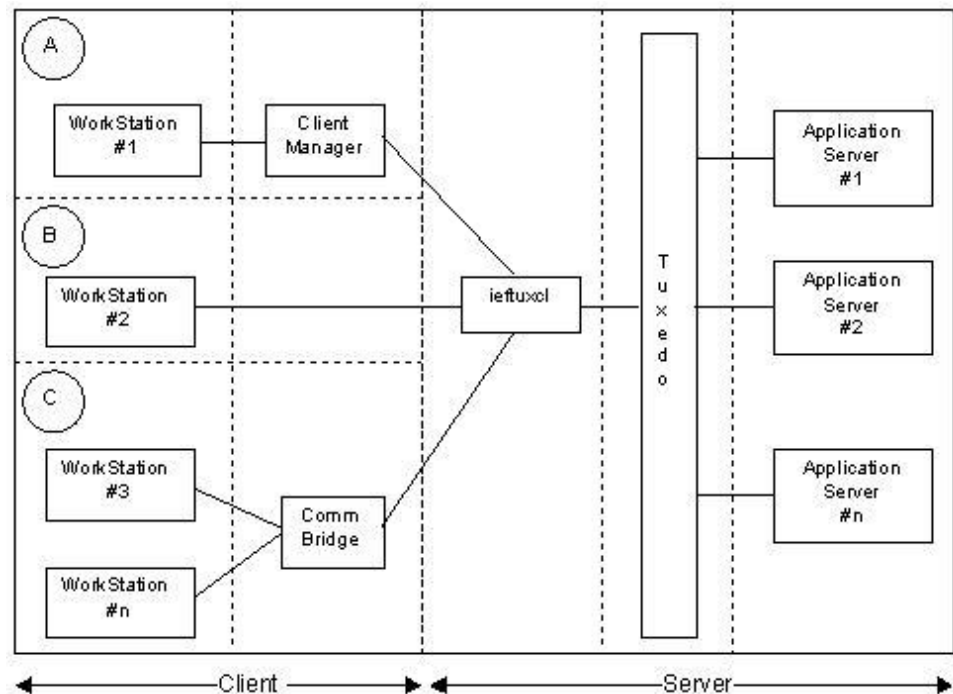
The CLASSPATH variable contains a colon separated list of jar files that are needed by the Java-based clients. Typically these are jolt.jar, joltadmin.jar, joltjse.jar, and so on.

Chapter 7: Tuxedo Proxy Client (ieftuxcl)

The CA Gen Tuxedo Proxy Client (ieftuxcl) functions as a Tuxedo client for multiple Gen client/server clients running on multiple workstations. It can receive TCP/IP connections from individual clients, the Client Manager, or the Communications Bridge (Comm Bridge). It routes requests to Tuxedo and responds back to the requesting client.

Each instance of ieftuxcl can handle up to 256 client applications, but operates as a single Tuxedo client. If necessary, multiple instances of ieftuxcl can be started.

The following diagram illustrates how a typical setup might be using the ieftuxcl.



Case A: Client generated with CA Gen as the Communication Type using Client Manager to communicate with the Tuxedo Application Servers through the ieftuxcl.

Case B: A client generated with TCP/IP as the Communication Type accessing the Tuxedo Application Servers through ieftuxcl.

Case C: Clients generated with TCP/IP as the Communication Type using the Communication (Comm) Bridge to communicate with the Tuxedo Application Servers through the ieftuxcl.

Startup Parameters

The following table lists the parameters that can be issued on the command line when starting ieftuxcl.

Parameter	Description
-i c	<p>This specifies a unique character used to construct a COMMID to identify the requesting application. Each instance of ieftuxcl must specify a different value.</p> <p>Example:</p> <pre>\$IEFH/bin/ieftuxcl -i A</pre> <p>It may be a good idea to push the process in the background. To determine the best way to do this, see the UNIX System guides.</p>
-l	<p>This specifies that each request should be verified by the security exit. The default is no security checking. If this switch is used, the security process will validate the user ID, password, and transaction name from each request.</p> <p>Note: If the CA Gen Security option is used, the buffer is not passed to this Security Exit.</p>

Environment Variables

The following table shows the environment variables and their descriptions.

Variable	Description
IEFH	This locates the CA Gen runtime libraries.
SERVICE_NAME	<p>This specifies the TCP/IP service used to accept connections from the Comm Bridge. The default is uxptcp. The name specified must be present in the /etc/services file. Each instance of ieftuxcl must use a unique service name and number.</p>
SECFILE	<p>This specifies the location of the exit user ID and password file of the default security. The default is secfile in the current working directory.</p> <p>This is an ASCII file containing a user ID and password on each line, separated by one or more blanks. The maximum length of each user ID and each password is 8 characters.</p> <p>If security checking is not requested using the -l startup parameter, this variable is not required.</p>

Stopping ieftuxcl

To stop ieftuxcl processes, add the following to the script to stop the Tuxedo configuration prior to the tmshutdown command:

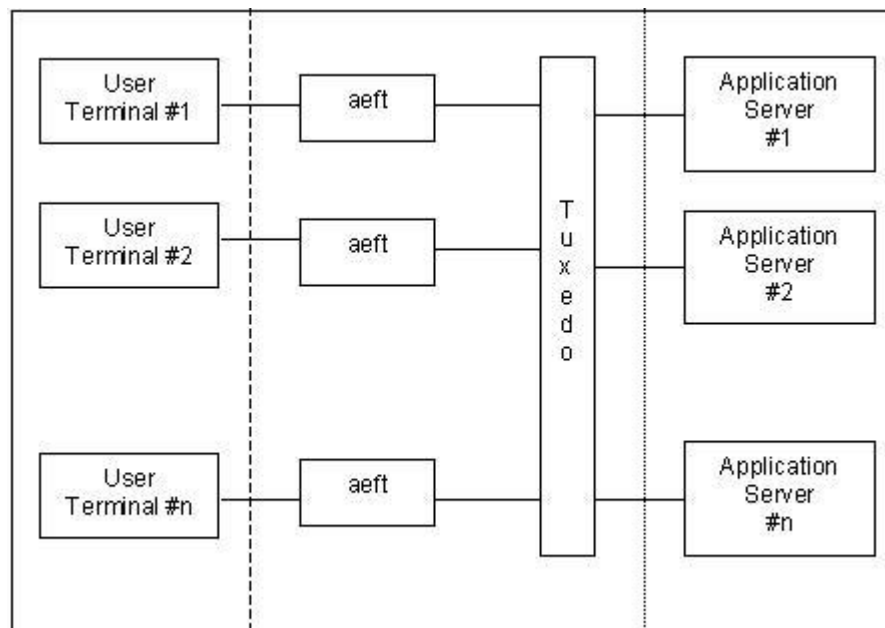
```
$ A='ps|grep ieftuxcl|cut -f1 -d"t"'
$ if test "$A" = ""
> then A='ps -e| grep ieftuxcl | cut -f1 -d"?"'
> fi
$ echo Stopping client ieftuxcl - pid $A
$ kill $A
```


Chapter 8: Block Mode Client (aef)

The AEF Tuxedo Client (aef) communicates with the Tuxedo and establishes a connection for the user. Each aef controls one user terminal.

The aef is a client program containing terminal screen handling logic that builds input message blocks and processes output message blocks. A CA Gen user on a local system starts this program. The aef connects the user to Tuxedo on the local system.

The aef is initially stored in location \$IEFH/bin, but it can be moved to another location.



Invoking the aef

The command syntax used to invoke the aef is:

```
aef [-s file_name] [-d number] [-p file_name] [-b] [-a] [-t number]
```

All options associated with the command are bracketed and defined in the following table.

Parameter	Description
-s <i>file_name</i>	The name of a script file created for the current session of input activity. This script file is used to replay the session at a later time.
-d <i>number</i>	The length of the delay between input messages when replaying a script file. The delay is measured in seconds. For example, -d 0 activates replay and with 0 seconds delay between message inputs. This parameter is <i>required</i> to activate the replay feature.
-p <i>file_name</i>	The name of a script file containing input activity saved during a previous session.
-b	Specifies that a blank screen be displayed during playback.
-a	Specifies that application menus control application security and that no Gen transaction screen is displayed. When conditions occur that cause the Gen transaction screen to appear, normal terminal operation terminates and playback is resumed.
-t <i>number</i>	A value between 0 and 31 used to activate an internal AEFT program trace (15 is most useful level). The option creates a file in the local directory containing the trace data. The filename is lg-aeft-<procid>.log where <procid> is the process number of the aeft (i.e. lg-aeft-163574.log).

Environment Variables

The following table lists aeft environment variables and descriptions.

Variable	Description
AEUSER [<i>unique, system-wide value</i>]	<p>Provides another way of using the same Gen application on multiple sessions on the same system.</p> <p>The AEUSER variable provides the unique key for Gen profile table access. Since Gen application restart depends on this key, careful consideration should be given to the method for building this unique value.</p> <p>The user ID field remains unchanged for screen display.</p>

Variable	Description
IEFH [directory_path]	Locates AEFT libraries. This variable must be set before an AEFT program is started.
AEPATH	<p>Search path for executable load modules.</p> <p>Set to a semicolon-delimited list of directory paths that indicate a search order for the user's application directory(s).</p> <p>Example:</p> <p>AEPATH=[directory_1;directory_2;...directory_n]</p> <p>Note: It is expected that each directory on the AEPATH contains an inqload directory where the executable load modules should reside.</p>

AEFT Command Example

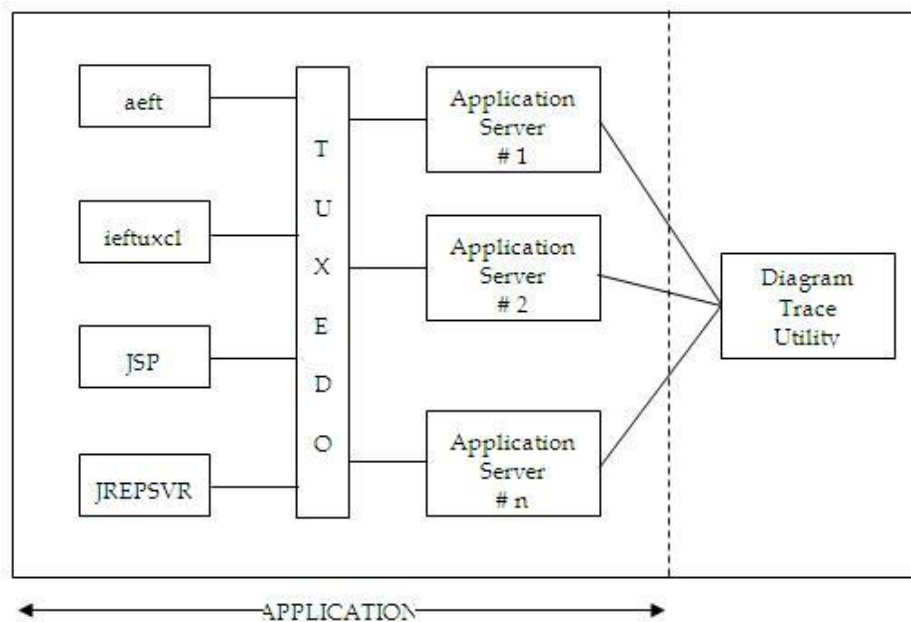
The following lines illustrate examples of commands that set certain environment variables and invoke the aeft. (These command lines assume you are using the C Shell.)

```
% setenv IEFH /iefrt
```

```
% aeft
```


Chapter 9: Testing Tuxedo Applications

Although an application is ready for execution after the remote files are processed, it is recommended that you test your application using the Diagram Trace Utility before introducing it into your production environment. The following illustration shows the communications between the various Tuxedo applications and the Diagram Trace Utility.



Using Diagram Trace Utility to Test Tuxedo Applications

The Diagram Trace Utility can be used to test a generated application before it is moved to a production environment.

Diagram Trace provides a number of special capabilities that allow viewing of the CA Gen model elements such as Action Diagram statements used to build the application while the program is being executed. To use the special capabilities of Diagram Trace, certain selections need to be made when remote files are generated. These selections cause the generation of the additional code required to implement the special capabilities available through Diagram Trace. Before a complete application can be tested, certain application components must be built using the Build tool:

- The application database
- RI trigger logic
- Any applicable operations libraries
- All load modules

Note: For more information on the Diagram Trace Utility, see the *Diagram Trace Utility User Guide*.

Regeneration for Changes

If changes need to be made to the application, the CA Gen model must be changed to generate the updated application, not the generated code. This applies whether the change is made to correct a problem or enhance the application. After the model is changed, regenerate that portion of the application on the development platform.

If the load module definition has not been updated by the changes, then only the necessary load module components need to be selected for regeneration instead of the complete model. The regenerated components can then be moved to the target system, overlaying the old versions of the components. The load module is then rebuilt. This saves the time and resources required to split a remote file whose definition has not changed.

Note: Make sure that file names and suffixes to file names are carefully verified before overlaying files. If file names differ, change the name of the new file to match the old one so that the MAKE command procedure used during the rebuild will search for the correct file.

If a load module definition has been changed, or if most of its components have been changed, then an entirely new remote file can be generated. The resulting remote file can then be moved to the target system, and the application built again.

After an application is tested, it can be regenerated without the trace code (if required) and installed for production use.

Application Testing

The following sections describe the concept of application testing and the procedure required to access and use the Diagram Trace Utility.

More information:

[Using the Application.ini File in Tuxedo](#) (see page 73)

Enabling Diagram Trace Utility

Follow these steps:

1. Build the test application that has been generated with trace.
2. Invoke the Diagram Trace Utility on any accessible Windows system (this can be the same system):
 - Launch the Diagram Trace Utility through the Start Menu: Start, All Programs, CA, Gen xx, Diagram Trace Utility.
Note: xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.
 - The Diagram Trace Utility will “autostart” and listen on port 4567. You may change the default port from within the Diagram Trace Utility.
3. Set the trace environment variables in the application.ini file.
4. Set your AEPATH environment variable to include the location of the Tuxedo application that you wish to test.

Accessing and Using Diagram Trace Utility

After the Diagram Trace Utility is enabled, invoke your application as you would normally.

When an application is being tested, any of its procedure steps and/or action blocks that have been generated with trace will communicate with the Diagram Trace Utility. No additional information is displayed with the application screens; the application will wait until control is returned from the Diagram Trace Utility.

Note: For more information, see the *CA Gen Diagram Trace Utility User Guide*.

Multi-user Diagram Trace Utility Support (for Tuxedo servers communicating with IEFTUXCL only)

When a remote server application is started, one copy of the application.ini file for that application is available. Setting the trace environment variables within this application.ini file will allow only one Diagram Trace Utility to trace any of the servers that make up this server application.

Test environments involve multiple testers working with the same server application. In this scenario, you need to debug from multiple client workstations and therefore, use multiple Diagram Trace Utilities.

More Information:

[Using the Application.ini File in Tuxedo](#) (see page 73)

Override the default trace environment variables

A client transaction can transfer the host and port of the client that is executing the transaction. By providing this information, the server can establish communication with the Diagram Trace Utility running on that client workstation. To trace the server for each transaction initiated from a client, use the client's Diagram Trace Utility.

Follow these steps:

1. Build the server application that has been generated with trace (as before).
2. Leave the trace environment variables within the application.ini file that is located in the server application's model directory commented out.
3. Start your servers using tmboot.
4. Invoke the Diagram Trace Utility on your client workstation.
5. Edit the application.ini file residing in your client application's executables directory.
6. Uncomment and set the trace environment variables. You can use 'localhost' for the TRACE_HOST variable.
7. Execute your client application.

The Diagram Trace Utility on the client workstation will be used to trace the server for each transaction initiated from that client.

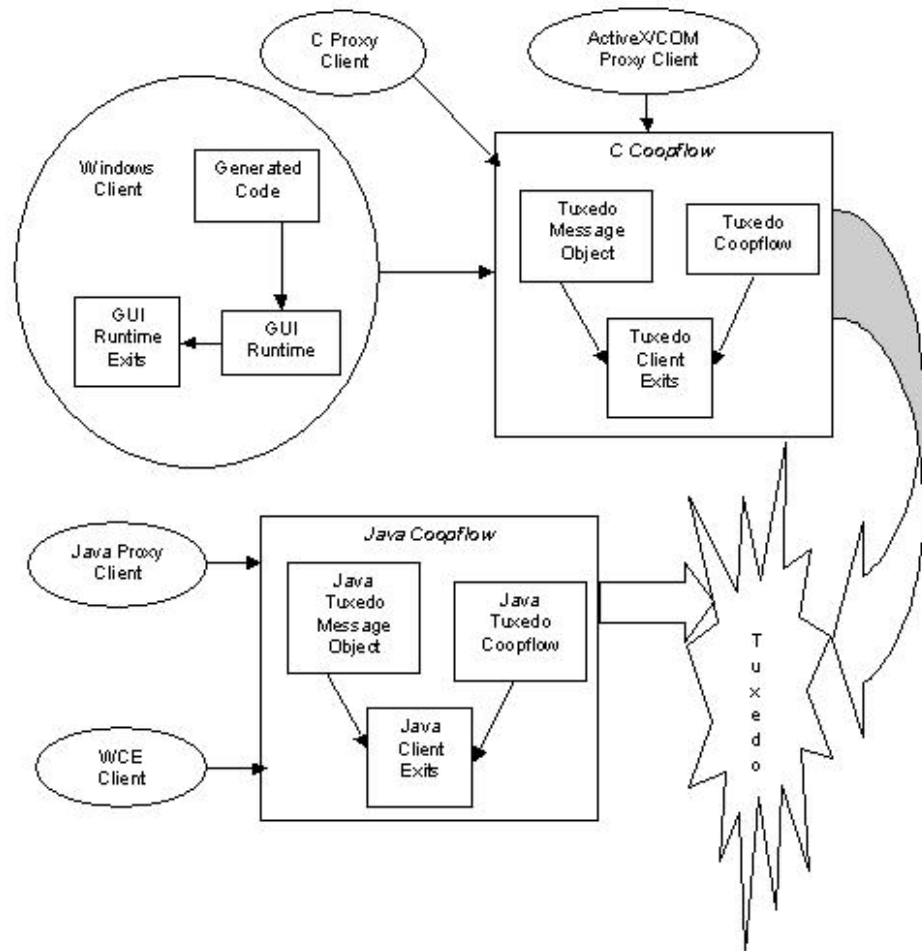
Regenerating Remote Files After Testing

The code generated on the CA Gen workstation for testing with trace includes features, such as trace calls, that are only needed during testing. These features increase the size of the load module significantly, as well as impact the application's performance. Even if no changes are required as a result of the tests performed, the load module should be regenerated without trace before it is implemented into production.

Chapter 10: Components of a Typical CA Gen Tuxedo System

Client-Side Components of a Typical CA Gen Tuxedo System

The following illustration shows the client-side components that are involved in processing a Distributed Processing cooperative flow to a Tuxedo application server.



Windows Client

Window Manager is code that is generated and built using CA Gen. This code provides the interface between the application and the user.

- **GUI Runtime**-Manages the Windows Client user interface and event handling for the application.
- **GUI Runtime Exits**-Provide opportunities to influence the processing of the GUI Runtime. The following GUI Runtime Exit entry points are of particular interest to Tuxedo users:
 - **WRSECTOKEN()**-Returns an indication that the system attributes CLIENT_USER_ID and CLIENT_PASSWORD should be made available to use in the connect processing of Tuxedo.
 - **WRSRVRError()**-This is given control when an error is received during the processing of a cooperative flow.

Note: For more information about the GUI runtime user exits, see the *User Exit Reference Guide*.

C Tuxedo Coopflow

The C Tuxedo Coopflow (Communications Runtime for Tuxedo) interfaces to the Tuxedo software using the ATMI API. This interface facilitates the client connect and disconnect from Tuxedo, performs Tuxedo buffer management, and performs the synchronous transmission of the cooperative flow to a Tuxedo service application (and so to the CA Gen Server Manager). The C Coopflow is made up of the following subcomponents:

- **C Tuxedo MsgObj**-Facilitates the transformation of data between the CA Gen import and export views and the View32 buffers of Tuxedo.
- **C Tuxedo CoopFlow**-Manages the interaction with the Tuxedo software using the ATMI API.

- **C Tuxedo Client Exits**-Provides exit opportunities to influence the processing of the C Tuxedo CoopFlow component. These exit points allow you to set the user context of a Tuxedo connection. Additionally, the data that is transferred can be viewed, modified, or both, prior to being sent to the server application and upon return from the server. The following Tuxedo Client Exit entry points are invoked by the C Tuxedo CoopFlow processing:
 - **ci_c_sec_set()**-Allows control of user context of Tuxedo connection.
 - **ci_c_user_data_out ()**-Provides access to outbound View32 buffer prior to invoking the target service (that is, before tpcall()).
 - **ci_c_user_data_in ()**-Provides access to inbound View32 buffer on return from the target service (that is, after tpcall()).

Note: For more information about the client user exits, see the *User Exit Reference Guide*.

Java Coopflow

The Java Coopflow (Java-based Communications Runtime for Tuxedo) interfaces to the Tuxedo software using the Java API from Tuxedo made available through the Jolt product of Oracle. This interface facilitates the client connect and disconnect from Tuxedo, performs Tuxedo buffer management, and performs the synchronous transmission of the cooperative flow to a Tuxedo service application (and so to the CA Gen Server Manager). The Java Coopflow is made up of the following subcomponents:

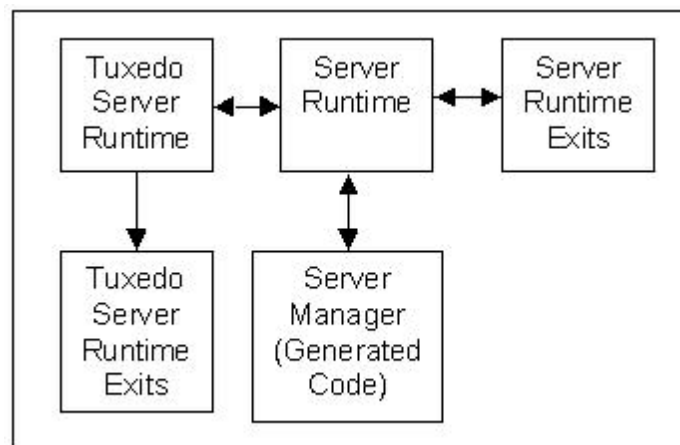
- **Java Tuxedo MsgObj**-The Java Tuxedo MsgObj sets up the message packets (Application import and export views) that are sent across to the Tuxedo Application Servers using the Java API of Jolt.
- **Java Tuxedo CoopFlow**-The Java Tuxedo Coopflow sets up the communications with the Tuxedo Application Servers using the Java API of Jolt.

- **Java Tuxedo Client Exits**-Provides exit opportunities to influence the processing of the Java CoopFlow. These exit points allow you to set the user context of a Jolt connection. Additionally, the data that is transferred can be viewed, modified, or both, prior to being sent to the server application and upon return from the server. The following Java Client Exits are invoked by the Java CoopFlow:
 - **TUXDynamicSecurityExit ()**-The Java Tuxedo CoopFlow instantiates an object of the security exit and provides it with the default User Name, User Password, Next Location, Tran Code, Procedure Name, and Model Name. The security exit can be modified to examine the supplied data and possibly modify the User Name and User Password. The security exit can also set the User Role and Application Password. The Java Tuxedo CoopFlow retrieves these parameters using the methods provided by the Security exit and uses them while creating a new Jolt Session using the Jolt API.
 - **TUXDynamicCoopFlowExit ()**-The Java Tuxedo CoopFlow instantiates an object of this user exit and provides it with a handle to the JoltRemoteService object. The OutData() member function can be modified to use the JoltRemoteService object to tailor the data that is sent to the server through the Java Tuxedo Message Object. The InData() member function can be modified to use JoltRemoteService to inspect the data received from the Tuxedo Application Server.

Note: For more information about the client user exits, see the *User Exit Reference Guide*.

Server-Side Components of a Typical CA Gen Tuxedo System

The following illustration shows the server-side components that are involved in processing a typical client-to-server Tuxedo cooperative flow.



Server Runtime

Provides flow control including the processing of server-to-server flows, error handling, mapping of view data and interactions with associated resource managers.

Server Runtime Exits

Provides opportunities to influence the processing of the server runtime. The following server Runtime Exit entry points are of particular interest to Tuxedo users:

- **TIRELOG ()**-Allows user access to error conditions that occur during the processing of a Server Manager Procedure Step. This exit is invoked from the Server Runtime of the Server Manager in which the error was encountered.
- **SRVERROR ()**-Invoked from the Server Runtime of the server that invoked the failing server, when processing a server-to-server flow. The exit provides access to the error returned from the failing Server Manager Procedure Step.

Tuxedo Server Runtime

Provides three entry points used by Tuxedo to pass control to the code of the service. These are:

- **tpsvrinit ()**-Called by Tuxedo when it boots a service.
- **TuxODCSvc ()**-Called by Tuxedo when it receives a cooperative flow buffer targeting a Procedure Step that corresponds to the associated Tuxedo service.
- **Tpsvrdone ()**-Called by Tuxedo when it shuts down a server.

Tuxedo Server Runtime Exits

Provide exit opportunities to influence the processing of the Tuxedo Server Runtime component. These exit points expose processing detail of the Tuxedo Server Runtime. Additionally, the data that is transferred can be viewed, modified, or both, upon receipt from the client and then prior to return to the client. The following Tuxedo Server Runtime Exit entry points are invoked by the Tuxedo Server Runtime processing:

- **ci_s_post_svrinit()**-Invoked at the very end of the processing of the tpsvrinit() function (see the above description of the Tuxedo Server Runtime).
- **ci_s_post_begin()**-Invoked when the flow, which is being processed by the Server, results in the initiation of a transaction.
- **ci_s_user_data_in()**-Invoked for each server providing access to the inbound View32 buffer being passed in to the server from the client application.
- **ci_s_user_data_out()**-Invoked for each server providing access to the View32 buffer being returned from the server to the invoking client application.
- **ci_s_pre_end()**-Invoked by the server that initiated a transaction, prior to terminating the global transaction.

- **ci_s_post_end()**-Invoked when the Server that initiated a transaction, prior to returning control to its invoking client, but after the associated global transaction has been completed (that is, committed or aborted).
- **ci_s_post_svrdone()**-Invoked at the very end of the processing of the tpsvrdone() function (see the previous description of the Tuxedo Server Runtime).

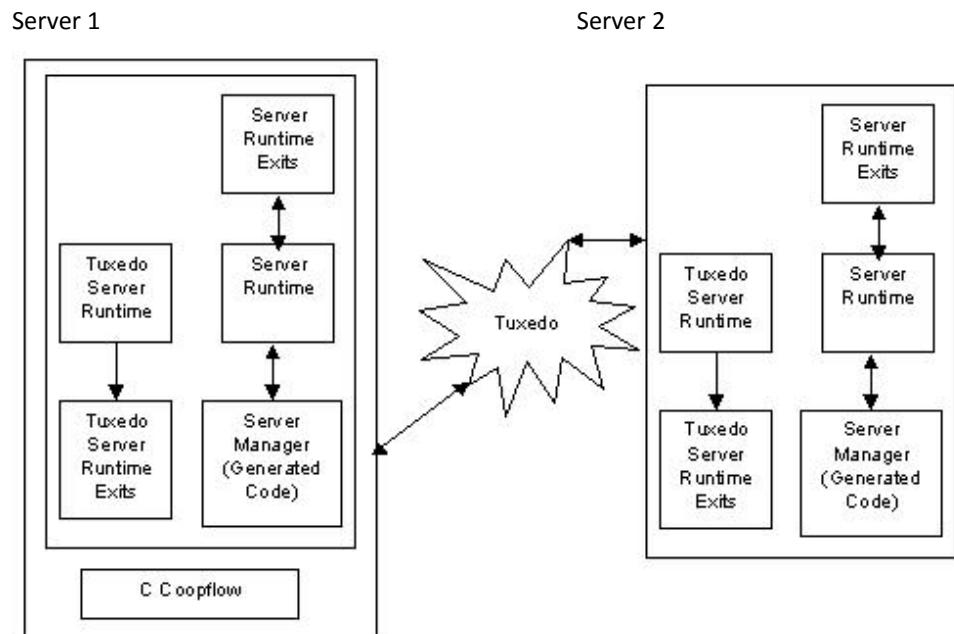
Note: For more information about the server runtime user exits, see the *User Exit Reference Guide*.

Server Manager

The Server Manager code is generated and built using CA Gen. The generated server code provides all the business logic and database access, and contains the Procedure Steps that are the targets of cooperative flow, that is, the clients request the services provided by a specific Procedure Step.

Components Involved in Server-to-Server Flow

The following illustration shows the components involved in processing a server-to-server Tuxedo cooperative flow.



This illustration shows the flow from one server manager to another server, hence the name server-to-server. The behavior of the server that initiates the flow is identical to that of the Windows client. A comparable component, Communications Runtime for Tuxedo, is added to the set of server components to process the flow initiated by the server that is now behaving as a client. The set of components of the target server remain the same as those that are involved in a client-to-server flow.

Chapter 11: User Exit Functions

The User Exit functions defined in this appendix are categorized according to the client or server component in which they are used.

Several of the User Exit Functions may be used by more than one component. For example, some of the client exits invoked for a Window Manager on a client-to-server flow are the same as those invoked from a Server Manager on a server-to-server flow.

Note: For a detailed description of each of the following user exits, see the “User Exits” appendix in the *Distributed Processing - Overview* Guide.

C/C++ Client Runtime Exits

The following user exits influence the processing within the GUI Runtime.

User Exit Name	Source Code	Description
WRSECTOKEN	wrexitn.c	Client-Side Security exit
WRSVRERROR	wrexitn.c	Sync Cooperative flow server error exit

The following user exits influence the processing within the C and COM Proxy Runtime.

User Exit Name	Source Code	Description
WRSECTOKEN	proxysit.c	Client-Side Security exit

The following user exits influence the behavior of the C Tuxedo CoopFlow processing.

User Exit Name	Source Code	Description
ci_c_sec_set	cictuxwsx.c	Set user supplied security data into the security data fields located in Tuxedo TPINIT structure.

User Exit Name	Source Code	Description
ci_c_user_data_out	cictuxwsx.c	Gives you the opportunity to inspect or modify the cooperative flow request buffer prior to Tuxedo sending the request to the target Tuxedo service. Invoked prior to the TPCALL for those clients connecting to Tuxedo servers residing on a separate host.
ci_c_user_data_in	cictuxwsx.c	Gives you the opportunity to inspect or modify the cooperative flow request buffer on return from the target Tuxedo service. Invoked on return from the TPCALL for those clients connecting to Tuxedo servers residing on a separate host. Additionally, this exit allows the client to disconnect from the server following each flow.
ci_c_user_data_out	cictuxx.c	Gives you the opportunity to inspect or modify the cooperative flow request buffer prior to Tuxedo sending the request to the target Tuxedo service. Invoked prior to the TPCALL for those clients connecting to Tuxedo servers residing on the same host (used for server-to-server flows).
ci_c_user_data_in	cictuxx.c	Gives you the opportunity to inspect or modify the cooperative flow request buffer on return from the target Tuxedo service. Invoked on return from the TPCALL for those clients connecting to Tuxedo servers residing on the same host. Additionally, this exit allows the client to disconnect from the server following each flow (used for server-to-server flows).

The following user exits influence the behavior of the Java Tuxedo CoopFlow processing.

Java Class Name	Description
TUXDynamicSecurityExit	The Java Tuxedo client security user exit

Java Class Name	Description
TUXDynamicCoopFlowExit	The Java Tuxedo Cooperative flow user exit

Server Runtime Exits

The server runtime exits provide opportunities to influence the processing of the server runtime. The two exits, TIRELOG and SRVRERROR, are only invoked if an error condition occurs.

User Exit Name	Source Code	Description
SRVRERROR	tirserrx.c	Server to server error exit that is invoked when an error condition occurs during server-to-server flow.
TIRELOG	tirelog.c	Server error handling routine invoked before the error response is returned to the client.

Tuxedo Server Runtime Exits

Tuxedo server runtime exits provide opportunities to access the request/reply buffer during the flow, as well as providing an opportunity to influence the Tuxedo server runtime component.

User Exit Name	Source Code	Description
ci_s_user_data_out	cistuxx.c	Gives you the opportunity to inspect or modify the cooperative flow request buffer prior to it being passed to the server manager.
ci_s_user_data_in	cistuxx.c	Gives you the opportunity to inspect or modify the cooperative flow request buffer on return from the server manager.
ci_s_post_begin	cistuxx.c	This user exit is called after Tuxedo tpbegin() is invoked.
ci_s_pre_end	cistuxx.c	This user exit is called just prior to a transaction being committed using Tuxedo tpcommit().

User Exit Name	Source Code	Description
ci_s_post_end	cistuxx.c	This user exit is called just after a transaction has been committed using Tuxedo tpcommit().
ci_s_post_svrinit	cistuxx.c	This user exit is called during server initialization by Tuxedo tmboot processing.
ci_s_post_svrdone	cistuxx.c	This user exit is called during server shutdown by Tuxedo tmshutdown processing.

Note: For more information about the server runtime user exits, see the *User Exit Reference Guide*.

Chapter 12: Error Messages

This appendix provides information on the types of messages output by CA Gen generated client and server programs in Tuxedo networks. The messages are presented in alphabetical order within message type.

If you receive any of these error messages, see the Tuxedo documentation at your site for an explanation of the cause of the error and the recovery procedures to use to resolve the problem.

Tuxedo errors are viewed from three distinct locations:

- ULOG.<MMDDYY> : This is the Tuxedo error logging file.
- CA Gen error message: Displayed to screen from CA Gen.
- stderr file : Generated by Tuxedo.

Client Error Messages

The following table explains the various client error messages.

Error	Remarks
The Tuxedo Client hangs while attempting a flow	<p>Tuxedo V6.3-the environment variables VIEWDIR32 and VIEWFILES32 on the server do not point to the view definition directory and files corresponding to the VIEW definitions of the client.</p> <p>Message in Tuxedo server ULOG file: 085245.nimrod!WSH.2830: WSNAT_CAT:1015: ERROR: Message decode failure 085245.nimrod!WSH.2830: WSNAT_CAT:1057: ERROR: Error processing message received from network (this is fixed in Tuxedo V6.4).</p>
TIRM030E: Application failed-Updates have been backed out	WSNADDR on client is not set or is set incorrectly.
ODC: Tuxedo Error in ci_c_sec_set = 12: TPESYSTEM - internal system error	Tuxedo is not available on the server machine (WSL is not running).

Error	Remarks
TIRM030E: Application failed-Updates have been backed out ODC: Tuxedo Error in ci_c_sec_set = 8: TPEPERM - bad permissions	Security is enabled for Tuxedo and the client did not give the valid application password or user ID and password combination.
TIRM030E: Application failed-Updates have been backed out ODC: Tuxedo Error in tpalloc(SERVER_IM) = 12: TPESYSTEM - internal system error	VIEWDIR32/VIEWFILES32 on client is not set or is set incorrectly.
TIRM030E: Application failed-Updates have been backed out ODC: Tuxedo Error in tpcall() = 6: TPENOENT - no entry found	The Tuxedo service is not available because it is unadvertised or because the server containing the service is no longer running.
TIRM030E: Application failed-Updates have been backed out ODC: Tuxedo Error in tpcall() = 12: TPESYSTEM - internal system error	Tuxedo V6.4-the VIEWDIR32/VIEWFILES32 environment variables do not contain the view definition for the client VIEW buffer. Tuxedo has been shutdown. (Need to issue a tpterm() on the client.)

Chapter 13: Flow Diagrams

This information is confidential. Log in to the CA Support site to access this information.

Follow these steps:

1. Go to <https://support.ca.com/irj/portal/DocumentationResults> and login.
2. Scroll down to Find Other Documentation area.
3. Type CA Gen in the product drop-down list.
4. Select Release 8.5 under Release and click go.
5. Click the PDF file for Tuxedo User Guide - Flow Diagrams.

The PDF file opens on your computer.

Note: Alternatively, you can search for the PDF using Q002931E as the document number.

Chapter 14: Using the Application.ini File in Tuxedo

Starting with AllFusion Gen 7.6, the runtime utilizes a file named application.ini. This file is delivered in the \$IEFH (for UNIX) directory. It contains a set of environment variables that are read by the runtime during application execution. The use of this file is to reduce the number of environment variables that must be set outside of the runtime for generated applications.

For providing environment variable uniqueness to each generated application, the application.ini file is copied from \$IEFH into the application's build directory. Before the application is executed, this file can be modified by the user to set these environment variables exclusively for the associated application. Once the file is copied into the application's build directory, it does not get overridden.

Application.ini Contents

The application.ini file contains a set of commented environment variables. Before using any of these environment variables, you must remove the comment from the variable (";") and modify the value as necessary.

The application.ini contains the following environment variables:

Environment Variable	Description	Default
TRACE_ENABLE	Used to enable communications with the Diagram Trace Utility.	1
TRACE_HOST	Used to define the Windows machine that will run the Diagram Trace Utility.	<none>
TRACE_PORT	Used to define the port that the Diagram Trace Utility will listen to.	4567

where TRACE_HOST can be in IPv4 or IPv6 format.

Index

A

- accessing Tuxedo servers
 - client-side changes • 42
 - server-side changes • 39
 - tasks before • 39
- adding WSL • 35
- aeft
 - command example • 49
 - use • 10
- application, testing • 53

B

- block mode client
 - environment variables • 48
 - invoking • 47
 - purpose • 47
- building C proxy clients
 - native • 38
 - workstation • 38

C

- C/C++ client runtime exits • 65
- client error messages • 69
- client-side changes
 - modifying commcfg properties file • 42
- client-side components
 - C Tuxedo overflow • 58
 - java Coopflow • 59
 - windows client • 58

D

- data dependent routing • 20
- designing and developing CA Gen applications for
 - data dependent routing • 20
 - failure handling • 19
 - retry • 19
 - transaction flows • 15
- detailing server managers for Tuxedo
 - invocation deadlock • 22
 - packaging issues • 21
 - setting individual server manager properties • 25
 - setting the business system server environment • 24

- diagram trace utility • 51

E

- enabling Tuxedo servers, for Java client access • 40
- ENVFILE, configuring on server machine • 30
- error messages, client • 69

F

- failure handling • 19

I

- ieftuxcl
 - as multiple Gen client/server clients • 43
 - stopping • 45
- individual server manager, properties setting • 25

J

- Jolt repository, bulk loading • 41

M

- Multi-User DTU support • 54

N

- native clients • 35
- Non-XA Tuxedo Servers • 18

P

- proxy client
 - environment variables • 44
 - startup parameters • 44
 - stopping • 45
- Pstep
 - failure handling • 19
 - invoking • 14
 - retry • 19

R

- Regenerating remote files after testing • 55
- Regeneration after testing • 52
- retry • 19
- RETRY_LIMIT, in transaction environment • 19
- routing, data dependent • 20
- runtime exits

- client • 65
- server • 67
- Tuxedo server • 67

S

- server runtime exits
 - SRVRERROR • 67
 - TIRELOG • 67
- server-side changes
 - adding JSL and JREPSVR • 40
 - bulk loading Jolt repository • 41
 - enabling Tuxedo servers for Java Client Access • 40
 - installing JSL and Jolt • 40
- server-side components
 - server manager • 62
 - server runtime • 61
 - Tuxedo server runtime • 61
- server-to-server flow, components involved • 62

T

- Testing
 - application • 53
 - Diagram trace • 51
 - Regenerating remote files after testing • 55
 - Regeneration after testing • 52
- TLOG file, creating • 34
- TMS, building • 27
- transaction
 - definition • 13
 - distributed • 13
 - flows • 15
- transaction control, using XA interface • 13
- transaction environment, avoid retrying in • 19
- Tuxedo
 - administrator tasks • 27
 - booting up the application servers • 34
 - detailing server managers for invocation
 - deadlock • 22
 - detailing server managers for packaging issues • 21
 - detailing the server managers • 23
 - distributed processing clients • 11
 - distributed processing server • 10
 - distributed transaction processing • 14
 - DP application development and deployment • 11
 - environment variables for client setup • 36

- error locations • 69
- invoking the aeft • 47
- non-XA servers • 18
- overview by chapter • 9
- prerequisites • 9
- proxy client • 43
- reference • 9
- related documentation • 9
- server runtime exits • 67
- server-side components • 60
- setting the business system server environment • 24
- setting the individual server manager properties • 25
- setting up server environment • 28
- support with CA Gen • 10
- transaction control with XA interface • 13
- Tuxedo system administration considerations
 - configure ENVFILE on the server • 30
 - configuring UBBCONFIG file • 30
 - create a transaction management server • 27
- Tuxedo system, client-side components
 - C Tuxedo coopflow • 58
 - Java coopflow • 59
 - windows client • 58
- two phase commit, process • 17

U

- UBBCONFIG file
 - compiling • 33
 - configuring • 30
 - sample • 30
 - setting TUXCONFIG value • 33
- user exit functions
 - C/C++ client runtime exits • 65
 - influencing the processing Java Tuxedo Coopflow • 65
 - influencing the processing of C Tuxedo Coopflow • 65
 - influencing the processing within the C and COM proxy runtime • 65
 - influencing the processing within the GUI runtime • 65
 - server runtime exits • 67
 - Tuxedo server runtime exits • 67

W

- workstation clients • 35

workstation listener, adding • 35

X

X/Open distributed transaction processing • 16