Using Oracle Forms with the Tuxedo TP Monitor

An Oracle White Paper

April 2000

ORACLE

Enabling the Information Age

TABLE OF CONTENTS

1. Introduction....1

1.1 Oracle Forms Developer....*1*

1.2 Oracle Forms Server....*2*

1.3 Tuxedo....*2*

2. The Interface....3

2.1 ATMI Interface....*3*

2.1.1 ATMI Constants and Structures....3

2.1.1.1 Flags to Service Routines....3

2.1.1.2 Flags to tpreturn()....4

2.1.1.3 Flags to tpscmt()....4

2.1.1.4 Flags to tpinit()....4

2.1.1.5 Flags to tpconvert()....4

2.1.1.6 Return Values from tpchkauth()....4

2.1.1.7 Maximum Length of a Tuxedo/T Identifier....5

2.1.1.8 tpinit() Interface Structure....5

2.1.1.9 Error Codes....5

2.1.1.10 Conversational and Event Flags....6

2.1.1.11 Queued Messages Add-on....6

2.1.1.12 Structure Elements that are Valid - Set in Flags....6

2.1.2 ATMI Functions....6

2.2 FML16 Interface....*10*

2.2.1 FML16 Constants and Structures....10

2.2.1.1 Constants....10

2.2.1.2 Operations for Fmodidx()....11

2.2.1.3 Flags for Fvstof()....11

2.2.1.4 Operations for Fstof....11

2.2.1.5 Field Types....11

2.2.1.6 Field Id Constants....11

2.2.1.7 Field Error Codes....12

2.2.2 FML16 Functions....12

2.2.2.1 Function Variants....12

2.2.2.2 Length Argument....13

2.2.2.3 Field Identifier Mapping Functions....13

2.2.2.4 Buffer Allocation and Initialization....14

2.2.2.5 Functions for Moving Fielded Buffers....14

2.2.2.6 Field Access and Modification....15

2.2.2.7 Buffer Update Functions....17

2.2.2.8 VIEWS Functions....17

2.2.2.9 Conversion Functions....18

2.2.2.10 Indexing Functions....20

2.2.2.11 Input/Output Functions....20

2.2.2.12 VIEW Conversion....21

2.2.2.13 Utility Functions....21

2.3 Additional Functions *21*
2.3.1 File I/O Functions 21
2.3.2 String Manipulation Functions 22
2.3.3 Shutdown Function 22

3. A Demonstration 23

3.1 Tuxedo bankapp *23*

3.2 Oracle Forms bankapp *24*
3.2.1 Preparing the bankapp Client 24
3.2.2 Running the bankapp Client 25

3.3 Client Development Tips *26*
3.3.1 Elements of the bankapp Client 26
3.3.1.1 bankapp Client PL/SQL Library 26
3.3.1.2 bankapp Client PL/SQL Form 30

4. Appendix 33

4.1 What's New in this Release? *33*
4.1.1 Bug Fixes 33
4.1.2 Current Limitations 33

4.2 Frequently Asked Questions *33*
4.2.1 General 33
4.2.2 Marketing 33

4.3 Additional Resources *34*
4.3.1 Oracle Forms 34
4.3.1.1 On-line Documentation 34
4.3.1.2 White Papers 34
4.3.1.3 Books 34
4.3.2 Tuxedo and TP Monitors 34
4.3.2.1 Documentation Set 34
4.3.2.2 White Papers 35
4.3.2.3 Books 35
4.3.2.4 Web Pages 35

1. Introduction

Oracle Forms Developer is an integrated tools suite for rapidly building sophisticated, complex, enterprise-class Internet applications for professional users. Oracle Forms Server is a complete application framework. It provides an extensible, optimized Java client; higher performance over any network; and out-of-the-box scalability for Web deployment.

Oracle Forms Server is the recommended solution for Web deployment of Oracle Forms applications. However, should you plan to deploy client/server applications in a three-tier transaction processing (TP) monitor architecture, this paper provides information on the use of Oracle Forms as a front-end development tool for the Tuxedo TP monitor. It also provides a brief introduction to Oracle Forms Developer and Oracle Forms Server, and describes and provides an example of the programmatic interface between Oracle Forms and Tuxedo, commonly referred to as D2TX.

The following table summarizes which releases of Oracle Developer can interface with which releases of Tuxedo.

Oracle Developer Release	Tuxedo Release
Developer/2000 1.3.2 for Windows 95/NT 3.51 Developer/2000 1.5.x for Windows 95/NT 3.51 Developer/2000 1.6 for Windows 95/NT 3.51 Developer/2000 2.x for Windows 95/NT 3.51	Tuxedo 6.1 volume 2 Part Number: 701-001004-001 (CD)
Developer/2000 1.3.3 for Windows 3.11 Developer Release 6i for Windows 95/NT 4.0 Developer Release 6i for Solaris 2.5.1	Tuxedo 6.4 Part Number: 701-001002-005 (CD)

Table 1 - Oracle Developer / Tuxedo Release Compatibility Matrix

1.1 Oracle Forms Developer

Oracle Forms Developer is a productive development environment for building enterprise-class Internet database applications. Use Forms Developer to rapidly build sophisticated applications for viewing, changing, and adding information to your database. Forms Developer provides a set of integrated builders that enable business developers to construct sophisticated database forms, charts, and business logic rapidly with minimal effort. The Forms Developer application development environment provides powerful declarative features, such as wizards, built-ins, and drag-and-drop, to enable business developers to create fully functional applications from database definitions with minimal coding in record time.

1.2 Oracle Forms Server

Oracle Forms Server is an optimized application server for deploying new and existing Oracle Forms Developer applications to the Internet. Forms Server delivers the application infrastructure and the event model to ensure that Internet-based applications automatically scale and perform over any network. Forms Server built-in services include transaction management, record caching, record locking, exception handling, and load balancing—all provided as part of the Forms Server engine.

> *Note: Refer to Oracle Technology Network (http://technet.oracle.com/) for additional white papers on Oracle Forms Server.*

1.3 Tuxedo

Tuxedo is a transaction processing (TP) monitor that is available for UNIX, Netware, and Windows NT operating systems on over thirty-five server hardware platforms. It supports Macintosh, OS/2, and Windows operating systems as client platforms.

2. The Interface

Business developers can use Oracle Forms to build client/server applications against relational databases. Tuxedo provides a public application programming interface (API) that allows developers to write client/server applications based on their TP monitor software.

This interface presents the Tuxedo client API as PL/SQL functions and procedures, so that Oracle Forms developers can create Tuxedo clients using PL/SQL. The PL/SQL library that contains the equivalents of the Tuxedo API for the 32-bit Windows platform is called D2TX. D2TX registers the Tuxedo client API as PL/SQL foreign functions, allowing the API to be accessed directly from within PL/SQL code.

While Tuxedo's public API is quite extensive, this version of the interface focuses only on functions a client program would utilize. Specifically, this interface is an encapsulation of Tuxedo's Application-to-Transaction Manager Interface (ATMI) API, and the 16-bit version of the Forms Manipulation Language API (FML16, or just FML). The details of exactly which constants, procedures, and functions have been exposed in Oracle Forms are presented below.

2.1 ATMI Interface

The following tables show those elements of the Tuxedo ATMI interface that are exposed in Oracle Forms.

2.1.1 ATMI Constants and Structures

The following tables indicate the mapping of C programming constructs in the Tuxedo header file *atmi.h* to their equivalent definitions in the PL/SQL package "TUXDEF".

2.1.1.1 Flags to Service Routines

"C" Constant		PL/SQL Equivalent	
`#define TPNOBLOCK`	`0x00000001`	`tuxdef.TPNOBLOCK`	`integer := 1`
`#define TPSIGSTRT`	`0x00000002`	`tuxdef.TPSIGSTRT`	`integer := 2`
`#define TPNOREPLY`	`0x00000004`	`tuxdef.TPNOREPLY`	`integer := 4`
`#define TPNOTRAN`	`0x00000008`	`tuxdef.TPNOTRAN`	`integer := 8`
`#define TPTRAN`	`0x00000010`	`tuxdef.TPTRAN`	`integer := 16`
`#define TPNOTIME`	`0x00000020`	`tuxdef.TPNOTIME`	`integer := 32`
`#define TPABSOLUTE`	`0x00000040`	`tuxdef.TPABSOLUTE`	`integer := 64`
`#define TPGETANY`	`0x00000080`	`tuxdef.TPGETANY`	`integer := 128`
`#define TPNOCHANGE`	`0x00000100`	`tuxdef.TPNOCHANGE`	`integer := 256`
`#define TPCONV`	`0x00000400`	`tuxdef.TPCONV`	`integer := 1024`
`#define TPSENDONLY`	`0x00000800`	`tuxdef.TPSENDONLY`	`integer := 2048`
`#define TPRECVONLY`	`0x00001000`	`tuxdef.TPRECVONLY`	`integer := 4096`
`#define TPACK`	`0x00002000`	`tuxdef.TPACK`	`integer := 8192`

2.1.1.2 Flags to tpreturn()

"C" Constant		PL/SQL Equivalent	
#define TPFAIL	0x00000001	tuxdef.TPFAIL	integer := 1
#define TPSUCCESS	0x00000002	tuxdef.TPSUCCESS	integer := 2
#define TPEXIT	0x08000000	tuxdef.TPEXIT	integer := 134217728

2.1.1.3 Flags to tpscmt()

"C" Constant		PL/SQL Equivalent	
#define TP_CMT_LOGGED	0x01	tuxdef.TP_CMT_LOGGED	integer := 1
#define TP_COMT_COMPLETE	0x02	tuxdef.TP_CMT_COMPLETE	integer := 2

2.1.1.4 Flags to tpinit()

"C" Constant		PL/SQL Equivalent	
#define TPU_MASK	0x00000007	tuxdef.TPU_MASK	integer := 7
#define TPU_SIG	0x00000001	tuxdef.TPU_SIG	integer := 1
#define TPU_DIP	0x00000002	tuxdef.TPU_DIP	integer := 2
#define TPU_IGN	0x00000004	tuxdef.TPU_IGN	integer := 4
#define TPSA_FASTPATH	0x00000008	tuxdef.TPSA_FASTPATH	integer := 8
#define TPSA_PROTECTED	0x00000010	tuxdef.TPSA_PROTECTED	integer := 16

2.1.1.5 Flags to tpconvert()

"C" Constant		PL/SQL Equivalent	
#define TPTOSTRING	0x40000000	tuxdef.TPTOSTRING	integer := 1073741824
#define TPCONVCLTID	0x00000001	tuxdef.TPCONVCLTID	integer := 1
#define TPCONVTRANID	0x00000002	tuxdef.TPCONVTRANID	integer := 2
#define TPCONVXID	0x00000004	tuxdef.TPCONVXID	integer := 4
#define TPCONVMAXSTR	256	tuxdef.TPCONVMAXSTR	integer := 256

2.1.1.6 Return Values from tpchkauth()

"C" Constant		PL/SQL Equivalent	
#define TPNOAUTH	0	tuxdef.TPNOAUTH	integer := 0
#define TPSYSAUTH	1	tuxdef.TPSYSAUTH	integer := 1
#define TPAPPAUTH	2	tuxdef.TPAPPAUTH	integer := 2

2.1.1.7 Maximum Length of a Tuxedo/T Identifier

"C" Constant	PL/SQL Equivalent
`#define MAXTIDENT 30`	`tuxdef.MAXTIDENT integer := 30`

2.1.1.8 tpinit() Interface Structure

"C" Structure	PL/SQL Equivalent

`struct tpinfo_t {`
`  char usrname[MAXTIDENT+2];`
`  char cltname[MAXTIDENT+2];`
`  char passwd [MAXTIDENT+2];`
`  char grpname[MAXTIDENT+2];`
`  long flags;`
`  long datalen;`
`  long data;`
`};`
`typedef struct tpinfo_t TPINIT;`
	`type tuxdef.TPINIT is record (`
	`  usrname  VARCHAR2(30),`
	`  cltname  VARCHAR2(30),`
	`  passwd   VARCHAR2(30),`
	`  grpname  VARCHAR2(30),`
	`  flags    PLS_INTEGER,`
	`  datalen  PLS_INTEGER,`
	`  data     PLS_INTEGER`
	`);`

2.1.1.9 Error Codes

"C" Constant	PL/SQL Equivalent
`#define TPMINVAL 0`	`tuxdef.TPMINVAL integer := 0`
`#define TPEABORT 1`	`tuxdef.TPEABORT integer := 1`
`#define TPEBADDESC 2`	`tuxdef.TPEBADDESC integer := 2`
`#define TPEBLOCK 3`	`tuxdef.TPEBLOCK integer := 3`
`#define TPEINVAL 4`	`tuxdef.TPEINVAL integer := 4`
`#define TPELIMIT 5`	`tuxdef.TPELIMIT integer := 5`
`#define TPENOENT 6`	`tuxdef.TPENOENT integer := 6`
`#define TPEOS 7`	`tuxdef.TPEOS integer := 7`
`#define TPEPERM 8`	`tuxdef.TPEPERM integer := 8`
`#define TPEPROTO 9`	`tuxdef.TPEPROTO integer := 9`
`#define TPESVCERR 10`	`tuxdef.TPESVCERR integer := 10`
`#define TPESVCFAIL 11`	`tuxdef.TPESVCFAIL integer := 11`
`#define TPESYSTEM 12`	`tuxdef.TPESYSTEM integer := 12`
`#define TPETIME 13`	`tuxdef.TPETIME integer := 13`
`#define TPETRAN 14`	`tuxdef.TPETRAN integer := 14`
`#define TPGOTSIG 15`	`tuxdef.TPGOTSIG integer := 15`
`#define TPERMERR 16`	`tuxdef.TPERMERR integer := 16`
`#define TPEITYPE 17`	`tuxdef.TPEITYPE integer := 17`
`#define TPEOTYPE 18`	`tuxdef.TPEOTYPE integer := 18`
`#define TPERELEASE 19`	`tuxdef.TPERELEASE integer := 19`
`#define TPEHAZARD 20`	`tuxdef.TPEHAZARD integer := 20`
`#define TPEHEURISTIC 21`	`tuxdef.TPEHEURISTIC integer := 21`
`#define TPEEVENT 22`	`tuxdef.TPEEVENT integer := 22`
`#define TPEMATCH 23`	`tuxdef.TPEMATCH integer := 23`
`#define TPEDIAGNOSTIC 24`	`tuxdef.TPEDIAGNOSTIC integer := 24`
`#define TPEMIB 25`	`tuxdef.TPEMIB integer := 25`
`#define TPMAXVAL 26`	`tuxdef.TPMAXVAL integer := 26`

2.1.1.10 Conversational and Event Flags

"C" Constant	PL/SQL Equivalent
`#define TPEV_DISCONIM 0x0001`	`tuxdef.TPEV_DISCONIM integer := 1`
`#define TPEV_SVCERR 0x0002`	`tuxdef.TPEV_SVCERR integer := 2`
`#define TPEV_SVCFAIL 0x0004`	`tuxdef.TPEV_SVCFAIL integer := 4`
`#define TPEV_SVCSUCC 0x0008`	`tuxdef.TPEV_SVCSUCC integer := 8`
`#define TPEV_SENDONLY 0x0020`	`tuxdef.TPSA_SENDONLY integer := 32`

2.1.1.11 Queued Messages Add-on

"C" Constant	PL/SQL Equivalent
`#define TMQNAMELEN 15`	`tuxdef.TMQNAMELEN integer := 15`
`#define TMMSGIDLEN 32`	`tuxdef.TMMSGIDLEN integer := 32`
`#define TMCORRIDLEN 32`	`tuxdef.TMCORRIDLEN integer := 32`

2.1.1.12 Structure Elements that are Valid - Set in Flags

"C" Constant	PL/SQL Equivalent
`#define TPNOFLAGS 0x00000`	`tuxdef.TPNOFLAGS integer := 0`
`#define TPQCORRID 0x00001`	`tuxdef.TPQCORRID integer := 1`
`#define TPQFAILUREQ 0x00002`	`tuxdef.TPQFAILUREQ integer := 2`
`#define TPQBEFOREMSGID 0x00004`	`tuxdef.TPQBEFOREMSGID integer := 4`
`#define TPQGETBYMSGID 0x00008`	`tuxdef.TPQGETBYMSGID integer := 8`
`#define TPQMSGID 0x00010`	`tuxdef.TPQMSGID integer := 16`
`#define TPQPRIORITY 0x00020`	`tuxdef.TPQPRIORITY integer := 32`
`#define TPQTOP 0x00040`	`tuxdef.TPQTOP integer := 64`
`#define TPQWAIT 0x00080`	`tuxdef.TPQWAIT integer := 128`
`#define TPQREPLYQ 0x00100`	`tuxdef.TPQREPLYQ integer := 256`
`#define TPQTIME_ABS 0x00200`	`tuxdef.TPQTIME_ABS integer := 512`
`#define TPQTIME_REL 0x00400`	`tuxdef.TPQTIME_REL integer := 1024`
`#define TPQGETBYCORRID 0x00800`	`tuxdef.TPQGETBYCORRID integer := 2048`
`#define TPQPEEK 0x01000`	`tuxdef.TPQPEEK integer := 4096`

2.1.2 ATMI Functions

The following tables indicate the mapping of C function prototypes in the Tuxedo header file *atmi.h* to the equivalent functions and procedures in the PL/SQL package "ATMI".

These are the ATMI functions proper. They are presented here in alphabetical order.

"C" Function Prototype	PL/SQL Equivalent
`int tpabort ( long flags );`	`function ATMI.tpabort ( flags in PLS_INTEGER ) return PLS_INTEGER;`
`int tpacall ( char *svc, char *data, long len, long flags );`	`function ATMI.tpacall ( svc in out VARCHAR2, data in ORA_FFI.POINTERTYPE, len in PLS_INTEGER, flags in PLS_INTEGER ) return PLS_INTEGER;`

"C" Function Prototype	PL/SQL Equivalent

int **tpadvertise** (
  char *svcname,
  void (*func)(TPSVCINFO *)
);
	Not a Tuxedo client function.

char ***tpalloc** (
  char *type,
  char *subtype,
  long size
);
	function ATMI.tpalloc (
	  type in out VARCHAR2,
	  subtype in out VARCHAR2,
	  size in PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

int **tpbegin** (
  unsigned long timeout,
  long flags
);
	function ATMI.tpbegin (
	  timeout in PLS_INTEGER,
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

int **tpbroadcast** (
  char *lmid,
  char *usrname,
  char *cltname,
  char *data,
  long len,
  long flags
);
	function ATMI.tpbroadcast (
	  lmid in out VARCHAR2,
	  usrname in out VARCHAR2,
	  cltname in out VARCHAR2,
	  data in ORA_FFI.POINTERTYPE,
	  len in PLS_INTEGER,
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

int **tpcall** (
  char *svc,
  char *idata,
  long ilen,
  char **odata,
  long *olen,
  long flags
);
	function ATMI.tpcall (
	  svc in out VARCHAR2,
	  idata in ORA_FFI.POINTERTYPE,
	  ilen in PLS_INTEGER,
	  odata in out ORA_FFI.POINTERTYPE,
	  olen in out PLS_INTEGER,
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

int **tpcancel** (
  int cd
);
	function ATMI.tpcancel (
	  cd in PLS_INTEGER
	) return PLS_INTEGER;

int **tpchkauth** (void);	function ATMI.tpchkauth return PLS_INTEGER;
int **tpchkunsol** (void);	function ATMI.tpchkunsol return PLS_INTEGER;
int **tpclose** (void);	*Not a Tuxedo client function.*

int **tpcommit** (
  long flags
);
	function ATMI.tpcommit (
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

int **tpconnect** (
  char *svc,
  char *data,
  long len,
  long flags
);
	function ATMI.tpconnect (
	  svc in out VARCHAR2,
	  data in ORA_FFI.POINTERTYPE,
	  len in PLS_INTEGER,
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

int **tpconvert** (
  char *arg1,
  char *arg2,
  long arg3
);
	Planned for a future release.

int **tpdequeue** (
  char *qspace,
  char *qname,
  TPQCTL *ctl,
  char **data,
  long *len,
  long flags
);
	Planned for a future release.

int **tpdiscon** (
  int cd
);
	function ATMI.tpdiscon (
	  cd in PLS_INTEGER
	) return PLS_INTEGER;

"C" Function Prototype	PL/SQL Equivalent

int **tpenqueue** (
  char *qspace,
  char *qname,
  TPQCTL *ctl,
  char *data,
  long len,
  long flags
);
	Planned for a future release.

void **tpforward** (
  char *svc,
  char *data,
  long len,
  long flags
);
	Not a Tuxedo client function.

void **tpfree** (
  char *ptr
);
	procedure ATMI.tpfree (
	  ptr in ORA_FFI.POINTERTYPE
	);

int **tpgetlev** (void);	function ATMI.tpgetlev return PLS_INTEGER;

int **tpgetrply** (
  int *cd,
  char **data,
  long *len,
  long flags
);
	function ATMI.tpgetrply (
	  cd in out PLS_INTEGER,
	  data in out ORA_FFI.POINTERTYPE,
	  len in out PLS_INTEGER,
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

int **tpgprio** (void);	function ATMI.tpgprio return PLS_INTEGER;

int **tpinit** (
  TPINIT *tpinfo
);
	Use the first variant if there is variable length string data that needs to be forwarded to an application-specific authentication service. Note that the length of the variable length string data is calculated internally, and that if an error is encountered, the error code is returned in the argument tperrno*.*
	
	function ATMI.tpinit (
	  usrname in VARCHAR2,
	  cltname in VARCHAR2,
	  passwd in VARCHAR2,
	  grpname in VARCHAR2,
	  flags in PLS_INTEGER,
	  data in out VARCHAR2,
	  tperrno out PLS_INTEGER
	) RETURN PLS_INTEGER;
	
	function ATMI.tpinit (
	  tpinfo in TUXDEF.TPINIT
	) return PLS_INTEGER;

int **tpnotify** (
  CLIENTID *clientid,
  char *data,
  long len
  long flags
);
	Not a Tuxedo client function.

int **tpopen** (void);	*Not a Tuxedo client function.*

int **tppost** (
  char *eventname,
  char *data,
  long len,
  long flags
);
	Planned for a future release.

"C" Function Prototype	PL/SQL Equivalent

char ***tprealloc** (
  char *ptr,
  long size
);
	function ATMI.tprealloc (
	  ptr in ORA_FFI.POINTERTYPE,
	  size in PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

int **tprecv** (
  int cd,
  char **data,
  long *len,
  long flags,
  long *revent
);
	function ATMI.tprecv (
	  cd in PLS_INTEGER,
	  data in out ORA_FFI.POINTERTYPE,
	  len in out PLS_INTEGER,
	  flags in PLS_INTEGER,
	  revent in out PLS_INTEGER
	) return PLS_INTEGER;

int **tpresume** (
  TPTRANID *tranid,
  long flags
);
	Planned for a future release.

void **tpreturn** (
  int rval,
  long rcode,
  char *data,
  long len,
  long flags
);
	Not a Tuxedo client function.

int **tpscmt** (long flags);	*Planned for a future release.*

int **tpsend** (
  int cd,
  char *data,
  long len,
  long flags,
  long *revent
);
	function ATMI.tpsend (
	  cd in PLS_INTEGER,
	  data in ORA_FFI.POINTERTYPE,
	  len in PLS_INTEGER,
	  flags in PLS_INTEGER,
	  revent in out PLS_INTEGER
	) return PLS_INTEGER;

void **tpservice** (TPSVCINFO *svcinfo);	*Not a Tuxedo client function.*

void (***tpsetunsol** (void (*disp)
  (char *data,
  long len,
  long flags))) (
  char *data,
  long len,
  long flags
);
	Planned for a future release.

int **tpsprio** (
  int prio,
  long flags
);
	function ATMI.tpsprio (
	  prio in PLS_INTEGER,
	  flags in PLS_INTEGER
	) return PLS_INTEGER;

char ***tpstrerror** (
  int err
);
	function ATMI.tpstrerror (
	  err in PLS_INTEGER
	) return VARCHAR2;

int **tpsubscribe** (
  char *eventexpr,
  char *filter,
  TPEVCTL *ctl,
  long flags
);
	Planned for a future release.

int **tpsuspend** (
  TPTRANID *tranid,
  long flags
);
	Planned for a future release.

void **tpsvrdone** (void);	*Not a Tuxedo client function.*

int **tpsvrinit** (
  int argc,
  char **argv
);
	Not a Tuxedo client function.

"C" Function Prototype	PL/SQL Equivalent
int **tpterm** (void);	function ATMI.tpterm return PLS_INTEGER;

long **tptypes** (
  char *ptr,
  char *type,
  char *subtype
);
	function ATMI.tptypes (
	  ptr in ORA_FFI.POINTERTYPE,
	  type in out VARCHAR2,
	  subtype in out VARCHAR2
	) return PLS_INTEGER;

int **tpunadvertise** (char *svcname);	*Not a Tuxedo client function.*

int **tpunsubscribe** (
  long subscription,
  long flags
);
	Planned for a future release.

While the following functions are not technically ATMI functions, their prototypes are in the Tuxedo header file *atmi.h*.

"C" Function Prototype	PL/SQL Equivalent
int **gettperrno** (void);	function ATMI.gettperrno return PLS_INTEGER;
long **gettpurcode** (void);	function ATMI.gettpurcode return PLS_INTEGER;

char ***tuxgetenv** (
  char *name
);
	function D2TX.tuxgetenv (
	  name in VARCHAR2
	return VARCHAR2;

int **tuxputenv** (
  char *string
);
	function D2TX.tuxputenv
	  string in VARCHAR2
	return PLS_INTEGER;

int **tuxreadenv** (
  char *file,
  char *label
);
	function D2TX.tuxreadenv
	  file in out VARCHAR2,
	  label in out VARCHAR2
	return PLS_INTEGER;

2.2 FML16 Interface

The following tables show those elements of the Tuxedo FML16 interface that are exposed in Oracle Forms.

2.2.1 FML16 Constants and Structures

The following tables indicate the mapping of C programming constructs in the Tuxedo header file *fml.h* to their equivalent definitions in the PL/SQL package "TUXDEF".

2.2.1.1 Constants

"C" Constant	PL/SQL Equivalent
#define MAXFBLEN 0xfffc	tuxdef.MAXFBLEN integer := 65532
#define FSTDXINT 16	tuxdef.FSTDXINT integer := 16
#define FMAXNULLSIZE 2660	tuxdef.FMAXNULLSIZE integer := 2660
#define FVIEWCACHESIZE 10	tuxdef.MAXFBLEN integer := 10
#define FVIEWNAMESIZE 33	tuxdef.MAXFBLEN integer := 33

2.2.1.2 Operations for Fmodidx()

"C" Constant		PL/SQL Equivalent	
#define FADD	1	tuxdef.FADD	integer := 1
#define FMLMOD	2	tuxdef.FMLMOD	integer := 2
#define FDEL	3	tuxdef.FDEL	integer := 3

2.2.1.3 Flags for Fvstof()

"C" Constant		PL/SQL Equivalent	
#define F_OFF	0	tuxdef.F_OFF	integer := 0
#define F_OFFSET	1	tuxdef.F_OFFSET	integer := 1
#define F_SIZE	2	tuxdef.F_SIZE	integer := 2
#define F_PROP	4	tuxdef.F_PROP	integer := 4
#define F_FTOS	8	tuxdef.F_FTOS	integer := 8
#define F_STOF	16	tuxdef.F_STOF	integer := 16
#define F_BOTH	(F_STOF | F_FTOS)	tuxdef.F_BOTH	integer := 24
#define F_LENGTH	32	tuxdef.F_LENGTH	integer := 32
#define F_COUNT	64	tuxdef.F_COUNT	integer := 64
#define F_NONE	128	tuxdef.F_NONE	integer := 128

2.2.1.4 Operations for Fstof

"C" Constant		PL/SQL Equivalent	
#define FUPDATE	1	tuxdef.FUPDATE	integer := 1
#define FCONCAT	2	tuxdef.FCONCAT	integer := 2
#define FJOIN	3	tuxdef.FJOIN	integer := 3
#define FOJOIN	4	tuxdef.FOJOIN	integer := 4

2.2.1.5 Field Types

"C" Constant		PL/SQL Equivalent	
#define FLD_SHORT	0	tuxdef.FLD_SHORT	integer := 0
#define FLD_LONG	1	tuxdef.FLD_LONG	integer := 1
#define FLD_CHAR	2	tuxdef.FLD_CHAR	integer := 2
#define FLD_FLOAT	3	tuxdef.FLD_FLOAT	integer := 3
#define FLD_DOUBLE	4	tuxdef.FLD_DOUBLE	integer := 4
#define FLD_STRING	5	tuxdef.FLD_STRING	integer := 5
#define FLD_CARRAY	6	tuxdef.FLD_CARRAY	integer := 6

2.2.1.6 Field Id Constants

"C" Constant	PL/SQL Equivalent
`#define BADFLDID (FLDID)0`	`tuxdef.BADFLDID integer := 0`
`#define FIRSTFLDID (FLDID)0`	`tuxdef.FIRSTFLDID integer := 0`

2.2.1.7 Field Error Codes

"C" Constant	PL/SQL Equivalent
`#define FMINVAL 0`	`tuxdef.FMINVAL integer := 0`
`#define FALIGNERR 1`	`tuxdef.FALIGNERR integer := 1`
`#define FNOTFLD 2`	`tuxdef.FNOTFLD integer := 2`
`#define FNOSPACE 3`	`tuxdef.FNOSPACE integer := 3`
`#define FNOTPRES 4`	`tuxdef.FNOTPRES integer := 4`
`#define FBADFLD 5`	`tuxdef.FBADFLD integer := 5`
`#define FTYPERR 6`	`tuxdef.FTYPERR integer := 6`
`#define FEUNIX 7`	`tuxdef.FEUNIX integer := 7`
`#define FBADNAME 8`	`tuxdef.FBADNAME integer := 8`
`#define FMALLOC 9`	`tuxdef.FMALLOC integer := 9`
`#define FSYNTAX 10`	`tuxdef.FSYNTAX integer := 10`
`#define FFTOPEN 11`	`tuxdef.FFTOPEN integer := 11`
`#define FFTSYNTAX 12`	`tuxdef.FFTSYNTAX integer := 12`
`#define FEINVAL 13`	`tuxdef.FEINVAL integer := 13`
`#define FBADTBL 14`	`tuxdef.FBADTBL integer := 14`
`#define FBADVIEW 15`	`tuxdef.FBADVIEW integer := 15`
`#define FVFSYNTAX 16`	`tuxdef.FVFSYNTAX integer := 16`
`#define FVFOPEN 17`	`tuxdef.FVFOPEN integer := 17`
`#define FBADACM 18`	`tuxdef.FBADACM integer := 18`
`#define FNOCNAME 19`	`tuxdef.FNOCNAME integer := 19`
`#define FMAXVAL 20`	`tuxdef.FMAXVAL integer := 20`

2.2.2 FML16 Functions

The following tables indicate the mapping of C function prototypes in the Tuxedo header file *fml.h* to the equivalent functions and procedures in the various FML PL/SQL packages. They are presented in the order in which they appear in Chapter 5, "Field Manipulation Functions," of the *Tuxedo FML Guide*.

2.2.2.1 Function Variants

Some of these functions, for example `fml.fchg()`, are overloaded to support more than one variable type for the argument that corresponds to the value of the field. The following table indicates the appropriate use of PL/SQL variable types and overloaded functions based on the field's type, as specified in the Tuxedo field table file.

PL/SQL Variable Types	Tuxedo FML Field Types
NUMBER	short, long, float, double
VARCHAR2	char, string, carray

If the FML field type is short, long, float, or double, then use the PL/SQL variable type NUMBER and the corresponding variant of an overloaded FML function. If the FML field type is char, string,

or carray, then use the PL/SQL variable type VARCHAR2 and the corresponding variant of an overloaded FML function.

2.2.2.2 Length Argument

Some of these functions, for example `fml.fget()`, have an argument in which the length of the receiving buffer is specified. There are two cases to consider:

1. If the field value will be *returned* as the FML field type short, long, float, or double, then the input value of the length argument will be ignored. The actual length of the field value that was written to the receiving buffer (PL/SQL variable) is still returned after the function has executed.
2. If the field value will be *returned* as the FML field type string, char or carray, then two options are available to the PL/SQL programmer:
   - For the fastest response time, the input value of the length argument should be equal to the maximum length of the VARCHAR2 variable that will receive the field value. For example, if the variable that will receive the field value is declared as VARCHAR2(100), then "100" should be used as the input value to the length argument.

     The actual length of the field value that was written to the receiving buffer (PL/SQL variable) is still returned after the function has executed.
   - If the input value of the length argument is specified to be (PL/SQL) NULL, then the maximum length of the receiving buffer (PL/SQL variable) will be calculated, and, consequently, the function will take longer to execute. The algorithm to determine the maximum length of the receiving buffer (PL/SQL variable) has been optimized, and choosing this option may not have an adverse impact on performance; however, it will *always be slower* than specifying the length explicitly.

     The actual length of the field value that was written to the receiving buffer (PL/SQL variable) is still returned after the function has executed.

2.2.2.3 Field Identifier Mapping Functions

"C" Function Prototype	PL/SQL Equivalent

`FLDID Fldid (`
`  char  *name`
`);`
	`function FML.fldid (`
	`  name  in out VARCHAR2`
	`) return PLS_INTEGER;`

`FLDOCC Fldno (`
`  FLDID  fieldid`
`);`
	`function FML.fldno (`
	`  fieldid  in  PLS_INTEGER`
	`) return PLS_INTEGER;`

`int Fldtype (`
`  FLDID  fieldid`
`);`
	`function FML.fldtype (`
	`  fieldid  in  PLS_INTEGER`
	`) return PLS_INTEGER;`

`FLDID Fmkfldid (`
`  int  type,`
`  FLDID  num`
`);`
	`function FML.fmkfldid (`
	`  type  in  PLS_INTEGER,`
	`  num  in  PLS_INTEGER`
	`) return PLS_INTEGER;`

`char *Fname (`
`  FLDID  fieldid`
`);`
	`function FML.fname (`
	`  fieldid  in  PLS_INTEGER`
	`) return VARCHAR2;`

`char *Ftype (`
`  FLDID  fieldid`
`);`
	`function FML.ftype (`
	`  fieldid  in  PLS_INTEGER`
	`) return VARCHAR2;`

2.2.2.4 Buffer Allocation and Initialization

"C" Function Prototype	PL/SQL Equivalent

FBFR *Falloc (
  FLDOCC F,
  FLDLEN V
);
	function FML.falloc (
	  f in PLS_INTEGER,
	  v in PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

int Ffree (
  FBFR *fbfr
);
	function FML.ffree (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Finit (
  FBFR *fbfr,
  FLDLEN buflen
);
	function FML.finit (
	  fbfr in ORA_FFI.POINTERTYPE,
	  buflen in PLS_INTEGER)
	) return PLS_INTEGER;

long Fneeded (
  FLDOCC F,
  FLDLEN V
);
	function FML.fneeded (
	  f in PLS_INTEGER,
	  v in PLS_INTEGER
	) return PLS_INTEGER;

FBFR *Frealloc (
  FBFR *fbfr,
  FLDOCC nf,
  FLDLEN nv
);
	function FML.frealloc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  nf in PLS_INTEGER,
	  nv in PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

long Fsizeof (
  FBFR *fbfr
);
	function FML.fsizeof (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

long Funused (
  FBFR *fbfr
);
	function FML.funused (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

long Fused (
  FBFR *fbfr
);
	function FML.fused (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

2.2.2.5 Functions for Moving Fielded Buffers

"C" Function Prototype	PL/SQL Equivalent

int Fcpy (
  FBFR *dest,
  FBFR *src
);
	function FML.fcpy (
	  dest in ORA_FFI.POINTERTYPE,
	  src in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Fmove (
  char *dest,
  FBFR *src
);
	function FML.fmove (
	  dest in ORA_FFI.POINTERTYPE,
	  src in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

2.2.2.6 Field Access and Modification

"C" Function Prototype	PL/SQL Equivalent

int **Fadd** (
  FBFR *fbfr,
  FLDID fieldid,
  char *value,
  FLDLEN len
);
	function FML.fadd (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in out VARCHAR2,
	  len in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML.fadd (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in NUMBER,
	  len in PLS_INTEGER
	) return PLS_INTEGER;

int **Fappend** (
  FBFR *fbfr,
  FLDID fieldid,
  char *value,
  FLDLEN len
);
	Planned for a future release.

int **Fchg** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  char *value,
  FLDLEN len
);
	function FML.fchg (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in out VARCHAR2,
	  len in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML.fchg (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in NUMBER,
	  len in PLS_INTEGER
	) return PLS_INTEGER;

int **Fcmp** (
  FBFR *fbfr1,
  FBFR *fbfr2
);
	function FML.fcmp (
	  fbfr1 in ORA_FFI.POINTERTYPE,
	  fbfr2 in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int **Fdel** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc
):
	function FML.fdel (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER
	) return PLS_INTEGER;

int **Fdelall** (
  FBFR *fbfr,
  FLDID fieldid
):
	function FML.fdelall (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER
	) return PLS_INTEGER;

int **Fdelete** (
  FBFR *fbfr,
  FLDID *fieldid
);
	function FML.fdelete (
	  fbfr in out ORA_FFI.POINTERTYPE,
	  fieldid in out PLS_INTEGER
	) return PLS_INTEGER;

char ***Ffind** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  FLDLEN *len
);
	function FML.ffind (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  len in out PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

char ***Ffindlast** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC *oc,
  FLDLEN *len
);
	function FML.ffindlast (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in out PLS_INTEGER,
	  len in out PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

"C" Function Prototype	PL/SQL Equivalent

FLDOCC **Ffindocc** (
  FBFR *fbfr,
  FLDID fieldid,
  char *value,
  FLDLEN len
);
	function FML.ffindocc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in out VARCHAR2,
	  len in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML.ffindocc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in NUMBER,
	  len in PLS_INTEGER
	) return PLS_INTEGER;

int **Fget** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  char *value,
  FLDLEN *maxlen
);
	function FML.fget (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in out VARCHAR2,
	  maxlen in out PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML.fget (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in out NUMBER,
	  maxlen in out PLS_INTEGER
	) return PLS_INTEGER;

char *__Fgetalloc__ (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  FLDLEN *extralen
);
	function FML.fgetalloc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  extralen in out PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

int **Fgetlast** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC *oc,
  char *value,
  FLDLEN *maxlen
);
	function FML.fgetlast (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in out PLS_INTEGER,
	  value in out VARCHAR2,
	  maxlen in out PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML.fgetlast (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in out PLS_INTEGER,
	  value in NUMBER,
	  maxlen in out PLS_INTEGER
	) return PLS_INTEGER;

int **Fnext** (
  FBFR *fbfr,
  FLDID *fieldid,
  FLDOCC *oc,
  char *value,
  FLDLEN *len
);
	function FML.fnext (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in out PLS_INTEGER,
	  oc in out PLS_INTEGER,
	  value in ORA_FFI.POINTERTYPE,
	  len in out PLS_INTEGER
	) return PLS_INTEGER;

FLDOCC **Fnum** (
  FBFR *fbfr
);
	function FML.fnum (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

FLDOCC **Foccur** (
  FBFR *fbfr,
  FLDID *fieldid
);
	function FML.foccur (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER
	) return PLS_INTEGER;

int **Fpres** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc
);
	function FML.fpres (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER
	) return PLS_INTEGER;

"C" Function Prototype	PL/SQL Equivalent

long **Fvall** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc
);
	function FML.fvall
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER
	) return PLS_INTEGER;

char ***Fvals** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc
);
	function FML.fvals (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER
	) return VARCHAR2;

2.2.2.7 Buffer Update Functions

"C" Function Prototype	PL/SQL Equivalent

int **Fconcat** (
  FBFR *dest,
  FBFR *src
);
	function FML.fconcat (
	  dest in ORA_FFI.POINTERTYPE,
	  src in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int **Fjoin** (
  FBFR *dest,
  FBFR *src
);
	function FML.fjoin (
	  dest in ORA_FFI.POINTERTYPE,
	  src in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int **Fojoin** (
  FBFR *dest,
  FBFR *src
);
	function FML.fojoin (
	  dest in ORA_FFI.POINTERTYPE,
	  src in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int **Fproj** (
  FBFR *fbfr,
  FLDID *fieldid
);
	function FML.fproj (
	  fbfr in out ORA_FFI.POINTERTYPE,
	  fieldid in out PLS_INTEGER
	) return PLS_INTEGER;

int **Fprojcpy** (
  FBFR *dest,
  FBFR *src,
  FLDID *fieldid
);
	function FML.fprojcpy (
	  dest in out ORA_FFI.POINTERTYPE,
	  src in out ORA_FFI.POINTERTYPE,
	  fieldid in out PLS_INTEGER
	) return PLS_INTEGER;

int **Fupdate** (
  FBFR *dest,
  FBFR *src
);
	function FML.fupdate (
	  dest in ORA_FFI.POINTERTYPE,
	  src in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

2.2.2.8 VIEWS Functions

"C" Function Prototype	PL/SQL Equivalent

int **Fvftos** (
  FBFR *fbfr,
  char *cstruct,
  char *view
);
	function FML_VIEWS.fvftos (
	  fbfr in ORA_FFI.POINTERTYPE,
	  cstruct in ORA_FFI.POINTERTYPE,
	  view in out VARCHAR2
	) return PLS_INTEGER;

int **Fvnull** (
  char *cstruct,
  char *cname,
  FLDOCC oc,
  char *view
);
	function FML_VIEWS.fvnull (
	  cstruct in ORA_FFI.POINTERTYPE,
	  cname in out VARCHAR2,
	  oc in PLS_INTEGER,
	  view in out VARCHAR2
	) return PLS_INTEGER;

int **Fvopt** (
  char *cname,
  int option,
  char *view
);
	function FML_VIEWS.fvopt (
	  cname in out VARCHAR2,
	  option in PLS_INTEGER,
	  view in out VARCHAR2
	) return PLS_INTEGER;

int **Fvselinit** (
  char *cstruct,
  char *cname,
  char *view
);
	function FML_VIEWS.fvselinit (
	  cstruct in ORA_FFI.POINTERTYPE,
	  cname in out VARCHAR2,
	  view in out VARCHAR2
	) return PLS_INTEGER;

"C" Function Prototype	PL/SQL Equivalent

int **Fvsinit** (
  char *cstruct,
  char *view
);
	function FML_VIEWS.fvsinit (
	  cstruct in ORA_FFI.POINTERTYPE,
	  view in out VARCHAR2
	) return PLS_INTEGER;

int **Fvstof** (
  FBFR *fbfr,
  char *cstruct,
  int mode,
  char *view
);
	function FML_VIEWS.fvstof (
	  fbfr in ORA_FFI.POINTERTYPE,
	  cstruct in ORA_FFI.POINTERTYPE,
	  mode in PLS_INTEGER,
	  view in out VARCHAR2
	) return PLS_INTEGER;

2.2.2.9 Conversion Functions

"C" Function Prototype	PL/SQL Equivalent

int **CFadd** (
  FBFR *fbfr,
  FLDID fieldid,
  char *value,
  FLDLEN len,
  int type
);
	function FML_CONV1.cfadd (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in out VARCHAR2,
	  len in PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML_CONV1.cfadd (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in NUMBER,
	  len in PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;

int **CFchg** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  char *value,
  FLDLEN len,
  int type
);
	function FML_CONV1.cfchg (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in out VARCHAR2,
	  len in PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML_CONV1.cfchg (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in NUMBER,
	  len in PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;

char ***CFfind** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  FLDLEN *len,
  int type
);
	function FML_CONV1.cffind (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  len in out PLS_INTEGER,
	  type in PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

"C" Function Prototype	PL/SQL Equivalent

FLDOCC **CFfindocc** (
  FBFR *fbfr,
  FLDID fieldid,
  char *value,
  FLDLEN len
  int type
);
	function FML_CONV2.cffindocc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in out VARCHAR2,
	  len in PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML_CONV2.cffindocc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in NUMBER,
	  len in PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;

int **CFget** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  char *buf,
  FLDLEN *len,
  int type
);
	function FML_CONV2.cfget (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  buf in out VARCHAR2,
	  len in out PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;
	
	function FML_CONV2.cfget (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  buf in out NUMBER,
	  len in out PLS_INTEGER,
	  type in PLS_INTEGER
	) return PLS_INTEGER;

char *__CFgetalloc__ (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  int type,
  FLDLEN *extralen
);
	function FML_CONV2.cfgetalloc (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  int in PLS_INTEGER,
	  extralen in out PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

int **Fadds** (
  FBFR *fbfr,
  FLDID fieldid,
  char *value
);
	function FML_CONVSTR.fadds (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  value in out VARCHAR2
	) return PLS_INTEGER;

int **Fchgs** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  char *value
);
	function FML_CONVSTR.fchgs (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  value in out VARCHAR2
	) return PLS_INTEGER;

char *__Ffinds__ (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc
);
	function FML_CONVSTR.ffinds (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER
	) return VARCHAR2;

int **Fgets** (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  char *buf
);
	function FML_CONVSTR.fgets (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  buf in out VARCHAR2
	) return PLS_INTEGER;

"C" Function Prototype	PL/SQL Equivalent

char *Fgetsa (
  FBFR *fbfr,
  FLDID fieldid,
  FLDOCC oc,
  FLDLEN *extra
);
	function FML_CONVSTR.fgetsa (
	  fbfr in ORA_FFI.POINTERTYPE,
	  fieldid in PLS_INTEGER,
	  oc in PLS_INTEGER,
	  extra in out PLS_INTEGER
	) return VARCHAR2;

char *Ftypcvt (
  FLDLEN *tolen,
  int totype,
  char *fromval,
  int fromtype,
  FLDLEN fromlen
);
	function FML_UTIL.ftypcvt (
	  tolen in out PLS_INTEGER,
	  totype in PLS_INTEGER,
	  fromval in ORA_FFI.POINTERTYPE,
	  fromtype in PLS_INTEGER,
	  fromlen in PLS_INTEGER
	) return ORA_FFI.POINTERTYPE;

2.2.2.10 Indexing Functions

"C" Function Prototype	PL/SQL Equivalent

long Fidxused (
  FBFR *fbfr
);
	function FML_INDEX.fidxused (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Findex (
  FBFR *fbfr,
  FLDOCC intvl
);
	function FML_INDEX.findex (
	  fbfr in ORA_FFI.POINTERTYPE,
	  intvl in PLS_INTEGER
	) return PLS_INTEGER;

int Frstrindex (
  FBFR *fbfr,
  FLDOCC numidx
);
	function FML_INDEX.frstrindex (
	  fbfr in ORA_FFI.POINTERTYPE,
	  numidx in PLS_INTEGER
	) return PLS_INTEGER;

FLDOCC Funindex (
  FBFR *fbfr
);
	function FML_INDEX.funindex (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

2.2.2.11 Input/Output Functions

"C" Function Prototype	PL/SQL Equivalent

long Fchksum (
  FBFR *fbfr
);
	function FML_IO.fchksum (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Fextread (
  FBFR *fbfr,
  FILE *iop
);
	function FML_IO.fextread (
	  fbfr in ORA_FFI.POINTERTYPE,
	  iop in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Ffprint (
  FBFR *fbfr,
  FILE *iop
);
	function FML_IO.ffprint (
	  fbfr in ORA_FFI.POINTERTYPE,
	  iop in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Fprint (
  FBFR *fbfr
);
	function FML_IO.fprint (
	  fbfr in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

int Fread (
  FBFR *fbfr,
  FILE *iop
);
	function FML_IO.fread (
	  fbfr in ORA_FFI.POINTERTYPE,
	  iop in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

"C" Function Prototype	PL/SQL Equivalent

int **Fwrite** (
  FBFR *fbfr,
  FILE *iop
);
	function FML_IO.fwrite (
	  fbfr in ORA_FFI.POINTERTYPE,
	  iop in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

2.2.2.12 VIEW Conversion

"C" Function Prototype	PL/SQL Equivalent
int **Fcodeset** (char *translation_table);	*Planned for a future release.*

long **Fvstot** (
  char *cstruct,
  char *trecord,
  long treclen,
  char *viewname
);
	Planned for a future release.

long **Fvttos** (
  char *cstruct,
  char *trecord,
  char *viewname
);
	Planned for a future release.

2.2.2.13 Utility Functions

While the following two functions are not technically FML functions, their prototypes are in the Tuxedo header file *fml.h*.

"C" Function Prototype	PL/SQL Equivalent
int **getFerror** (void);	function FML_UTIL.getFerror return PLS_INTEGER;

char ***Fstrerror** (
  int err
);
	function FML_UTIL.fstrerror (
	  err in PLS_INTEGER
	) return VARCHAR2;

2.3 Additional Functions

Oracle provides several other functions that may prove useful when developing applications with this interface.

2.3.1 File I/O Functions

"C" Function Prototype	PL/SQL Equivalent

int **fclose** (
  FILE *stream
);
	function FML_IO.fclose (
	  stream in ORA_FFI.POINTERTYPE
	) return PLS_INTEGER;

"C" Function Prototype	PL/SQL Equivalent

`FILE *fopen (`
`  char  *filename,`
`  char  *mode`
`);`
	`function FML_IO.fopen (`
	`  filename  in out VARCHAR2,`
	`  mode  in out VARCHAR2`
	`) return ORA_FFI.POINTERTYPE;`

2.3.2 String Manipulation Functions

These functions may prove particularly useful when the Tuxedo application uses string buffers rather than the other buffer types. They can also be used whenever PL/SQL variables of type `ORA_FFI.POINTERTYPE` and `VARCHAR2` need to be converted from one type to the other.

"C" Function Prototype	PL/SQL Equivalent

None.	`function D2TX.getstr (`
	`  ptr  in  ORA_FFI.POINTERTYPE`
	`) return VARCHAR2;`

`char *strcpy (`
`  char  *dest,`
`  const char *src`
`);`
	`function D2TX.strcpy (`
	`  dest  in out VARCHAR2,`
	`  src  in  ORA_FFI.POINTERTYPE`
	`) return VARCHAR2;`
	
	`function D2TX.strcpy (`
	`  dest  in  ORA_FFI.POINTERTYPE,`
	`  src  in out VARCHAR2`
	`) return VARCHAR2;`

2.3.3 Shutdown Function

This function unloads the interface dynamic-link library (d2txnn.dll) or shared object (d2txnn.so). The recommended place to use it is in the form's POST-FORM trigger (see Section 3.3.1.2, "bankapp Client PL/SQL Form," below).

"C" Function Prototype	PL/SQL Equivalent
None.	`procedure D2TX.shutdown;`

3. A Demonstration

The Tuxedo product is shipped with an example bank application (bankapp) to act as a working example of a Tuxedo-based client/server system. To demonstrate the interface software, the bankapp client was rewritten in PL/SQL using Oracle Forms. Prior to running the Oracle Forms bankapp client, it is necessary to install, configure, and run the Tuxedo product and bankapp application.

This section briefly describes how to prepare and run the native Tuxedo bankapp application and the Oracle Forms bankapp client. It offers some guidelines for developing Tuxedo clients with Oracle Forms.

3.1 Tuxedo bankapp

While it is beyond the scope of this white paper to act as the definitive guide to the installation, configuration, and execution of the Tuxedo bankapp, the following steps are provided as a guide for those who are new to Tuxedo. You will find the detailed information necessary to successfully complete this process in Tuxedo's documentation. You may find further assistance from Tuxedo's technical support organization.

Briefly, the steps to install, configure, and execute the Tuxedo bankapp are:

1. Install the Tuxedo product software on the server machine.

   For more information, refer to the *BEA Tuxedo System 6 Installation Guide*. Pay particular attention to the section titled "Operating System Configuration."

2. Optionally, create a Tuxedo administration account on the server machine (although, in practice, just about any existing account will do). This account will be executing Tuxedo bankapp server software.

3. Build and run the simple application that Tuxedo provides (simpapp) to minimally verify the installation. In simpapp, a software client requests a simple service, and the service returns the result. Note that both of these programs execute on the server machine.

   This is described in the *BEA Tuxedo System 6 Installation Guide*, as well as Chapter 1 of the *TUXEDO Application Development Guide*.

4. Build and run the bank application that Tuxedo provides (bankapp) as a more sophisticated example of an application layered on top of Tuxedo. Both the client and server programs execute on the server machine.

   It is very likely that you will need to adjust the operating system tunable parameters. This means that you will also have to reboot the server machine. Refer to the *Tuxedo System 6 Installation Guide* for help with the tunable parameters.

   You can find additional information about bankapp in the *TUXEDO Application Development Guide*. There is another useful document, *Exploring TUXEDO Using the bankapp Demo Program*, written by C. Cash Perkins, and dated 12/7/95. The latter takes some of the mystery out of getting the bankapp programs to work.

   Once bankapp is up and running, create a new bank account and make some deposits and withdrawals. If you like, use the example account number "20020." You can use this account

later to verify that the Oracle Forms bankapp client works as well as the Tuxedo bankapp client.

5. Install the Tuxedo software on the client machine.

   This is fairly straightforward. See the *BEA Tuxedo System 6 Installation Guide* for more information. It is important that the Tuxedo libraries are accessible from the system path. To accomplish this on Windows95, add the following two lines in the file AUTOEXEC.BAT. The value of TUXDIR should reflect the path where Tuxedo was installed on the client machine.

   ```
   SET TUXDIR=C:\tuxedo\6.4\ws\win32
   SET PATH=%PATH%;%TUXDIR%\bin\
   ```

   For Windows NT 4.0, open the Control Panel, then open the System Properties dialog box and select the Environment tab to set these environment variables.

   On a UNIX operating system, this can be done with something like the following two lines of C shell code. Again, the value of TUXDIR should reflect the path where Tuxedo was installed on the client machine.

   ```
   setenv TUXDIR /tuxedo\6.4
   setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${TUXDIR}/lib
   ```

   Note: Failure to ensure that the Tuxedo dynamic-link libraries or shared objects are on the system path will result in the inability to open the D2TX dynamic-link library or shared object when running a form that uses D2TX.

6. Build and run the bankapp client software on the client machine. The bankapp client and bankapp server programs now run on different machines. This exercise verifies that there is connectivity between machines, and that the Tuxedo software has been installed correctly on both machines. Use the new bank account that you created earlier.

   In a sense, this last step is the crux of the process. The document *Exploring TUXEDO Using the bankapp Demo Program* is very useful here, as are the log files, in the event that the bankapp client does not work correctly. A call to Tuxedo technical support might also be necessary.

3.2 Oracle Forms bankapp

Once the native Tuxedo bankapp is up and running correctly over the network, on separate client and server machines, the interface software (D2TX) can be demonstrated by running the Oracle Forms bankapp client.

3.2.1 Preparing the bankapp Client

The following table contains the names of the files that are appropriate for this version of the interface.

File Name	Description
`bankapp.fmb`	*bankapp form module binary file* - This is the Oracle Forms bankapp client.
`bankapp.pll`	*bankapp PL/SQL library module binary file* - This contains the bankapp utilities, and the abstraction of the bankapp client services, written in PL/SQL.

File Name	Description
d2tx.pll	*Oracle Forms - Tuxedo PL/SQL library module binary file* - This contains the PL/SQL versions of the Tuxedo program elements (ATMI and FML16 APIs) that are exposed in Oracle Forms.
d2txnn.dll	*Oracle Forms - Tuxedo dynamic-link library* - This contains those Tuxedo program elements that could not be encapsulated directly in PL/SQL. This file is automatically installed in the %ORACLE_HOME%\bin directory for the 32-bit Windows OS platforms.
d2txnn.so	*Oracle Forms - Tuxedo shared object* - This contains those Tuxedo program elements that could not be encapsulated directly in PL/SQL. This file is automatically installed in the ${ORACLE_HOME}\bin directory for the UNIX OS platforms.

Table 2 - Descriptions of Product Files

To prepare the Oracle Forms bankapp client, use the Oracle Installer to install the interface software (Oracle Forms Open Interfaces → Tuxedo Interface).

3.2.2 Running the bankapp Client

Assuming that Oracle Forms and D2TX have been successfully installed on the client machine, take the following steps to run the Oracle Forms bankapp client on the client machine:

1. Make sure that the correct version of Tuxedo/Workstation (/WS) is installed on the client machine. This is specified in the table in Section 1, "Introduction," on page 1.
2. Verify that the Tuxedo libraries (DLLs or shared objects) are accessible from the system path. Refer to Step 5 of Section 3.1, "Tuxedo bankapp," on page 23 for more details.
3. Ensure that the Tuxedo bankapp servers are running on the server machine. Ideally, they haven't been shut down since Tuxedo bankapp client was last run. Please refer to Step 4 of Section 3.1, "Tuxedo bankapp," on page 23 for more details.
4. On the 32-bit Windows OS platforms, open the Form Builder and run the bankapp Form module binary file (BANKAPP.FMB).

   On a UNIX OS platform, run the C shell script "fbankapp". This will automatically run the Oracle Forms bankapp client.
5. When the form (Oracle Forms bankapp client) comes up, click the Connect button.

   This is the most likely point where any problems with the demo may arise. These could include the inability to find the D2TX DLL or shared objects, or the inability to communicate with the Tuxedo bankapp servers. These problems will be displayed in the Forms message line, and logged in the file D2TX_ERR.LOG in the Form Builder or Forms Runtime working directory. You can minimize potential problems by making sure in advance that the native Tuxedo bankapp client can run successfully on the client machine.
6. Once the connection has been made, it is possible to use the radio buttons in the right-hand pane to process transactions. This should behave just as the native Tuxedo bankapp client did, except that now, it's implemented as an Oracle form. This is a good time to use the new account number that you created earlier.
7. Press the button labeled "Exit" to leave the Oracle Forms bankapp client.

3.3 Client Development Tips

This section offers some tips for developing Tuxedo clients with Oracle Forms, using the Oracle Forms bankapp client as an example.

3.3.1 Elements of the bankapp Client

Before delving into the specific tips, become familiar with the elements of the Oracle Forms bankapp client. The easiest way to do this is to load it into the Form Builder, keeping in mind that there are two sets of source code that will be reviewed: the source code associated with the Oracle Forms bankapp client *form*, and the source code that resides in the Oracle Forms bankapp client *PL/SQL library*.

3.3.1.1 bankapp Client PL/SQL Library

To take a look at the source code in the bankapp client PL/SQL library, start Form Builder and open the PL/SQL library, BANKAPP.PLL. Expand the Program Units to find the following PL/SQL program units:

PL/SQL Program Unit	Description
`BANKDEF (Package Spec)`	Defines exceptions and variables for global use.
`BANKSVCS (Package Spec)`	Specifies the interface for the BANKSVCS PL/SQL package, which comprises eight application-level services.
`BANKSVCS (Package Body)`	Implements the previous specification. These bank services are built on top of the bank utilities that are provided in the PL/SQL package BANKUTL.
`BANKUTL (Package Spec)`	Specifies the interface for the BANKUTL PL/SQL package, which comprises ten bank utility procedures and functions.
`BANKUTL (Package Body)`	Implements the previous specification. These bank utilities are built on top of the ATMI and FML PL/SQL packages that compose the Oracle Forms - Tuxedo PL/SQL library (D2TX.PLL).

Table 3 - Descriptions of bankapp PL/SQL Library Program Units

Note that the bankapp client is implemented in *layers*. The bankapp client (form) is built on top of the bank services; the bank services are built on top of the bank utilities; and the bank utilities are built on top of the PL/SQL versions of the Tuxedo client program elements (ATMI and FML16 APIs). Figure 3 illustrates the layers that are involved

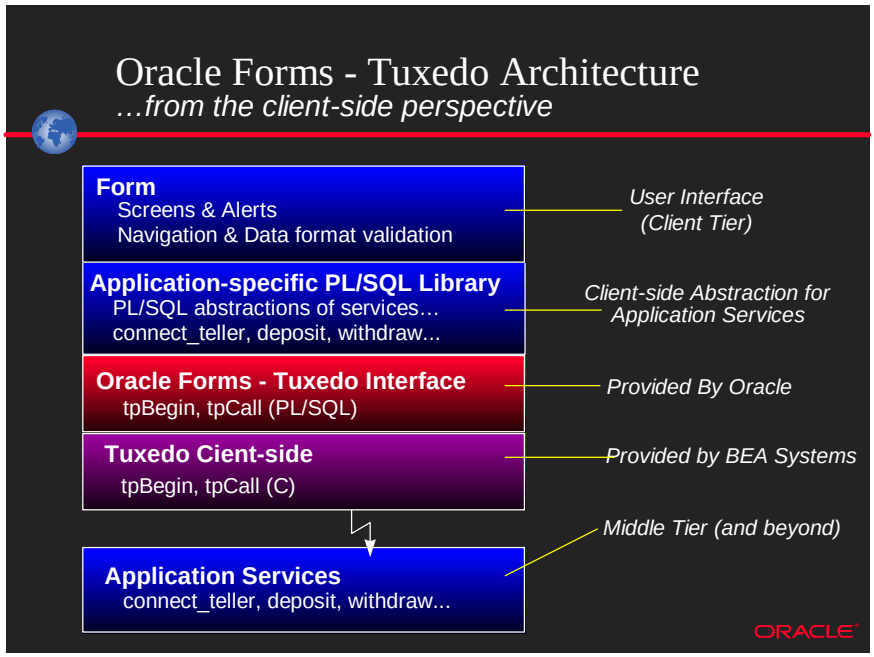

Figure 3 - Oracle Forms - Tuxedo Client Architecture

Alternatively, the Bank Services can be appreciated in their programmatic form. Below is the corresponding PL/SQL package specification. Note that the functions reflect some of the bank's business activities.

```
package BANKSVCS is

-- Copyright (C) Oracle Corporation 1996, 1998.
-- All Rights Reserved, Worldwide.


  procedure CONNECT_TELLER (errmsg in out varchar2);

  procedure DISCONNECT (errmsg in out varchar2);

  procedure INQUIRY (account_id in out pls_integer,
                     balance    in out number,
                     errmsg     in out varchar2);

  procedure DEPOSIT (account_id in out pls_integer,
                     amount     in out number,
                     balance    in out number,
                     errmsg     in out varchar2);

  procedure WITHDRAW (account_id in out pls_integer,
                      amount     in out number,
                      balance    in out number,
                      errmsg     in out varchar2);

  procedure TRANSFER (from_acct in out pls_integer,
                      to_acct   in out pls_integer,
                      amount    in out number,
                      from_bal  in out number,
                      to_bal    in out number,
                      errmsg    in out varchar2);

  procedure OPEN (lastname    in out varchar2,
                  firstname   in out varchar2,
                  midinitial  in out varchar2,
                  address     in out varchar2,
                  ssn         in out varchar2,
                  phone       in out varchar2,
                  initbalance in out number,
                  accttype    in out varchar2,
                  branchid    in out pls_integer,
```

```
                  account_id  in out pls_integer,
                  openbalance in out number,
                  errmsg      in out varchar2);

  procedure CLOSE (account_id in out pls_integer,
                   balance    in out number,
                   errmsg     in out varchar2);
end;
```

Similarly, the Bank Utilities are presented below in their PL/SQL package specification form.

```
package BANKUTL is

-- Copyright (C) Oracle Corporation 1996, 1998.
-- All Rights Reserved, Worldwide.

  procedure COMPOSE_ERROR (fbfr in out ora_ffi.pointertype,
                           errmsg in out varchar2);

  function ALLOC_MEM (memtyp in varchar2,
                      memsize in pls_integer) return ora_ffi.pointertype;

  procedure FREE_MEM (pointer in out ora_ffi.pointertype);

  procedure SET_VALUE (fbfr     in out ora_ffi.pointertype,
                       fldname  in out varchar2,
                       instance in     pls_integer,
                       value    in out pls_integer);

  procedure SET_VALUE (fbfr     in out ora_ffi.pointertype,
                       fldname  in out varchar2,
                       instance in     pls_integer,
                       value    in out varchar2);

  function GET_DOLLAR (fbfr     in out ora_ffi.pointertype,
                       fname    in out varchar2,
                       instance in     pls_integer) return number;

  function GET_NUMBER (fbfr in out ora_ffi.pointertype,
                       fname in out varchar2,
                       instance in pls_integer) return number;

  function CALL_SERVICE (svcname in out varchar2,
                         fbfr    in out ora_ffi.pointertype,
                         buflen  in out pls_integer) return pls_integer;

  procedure BEGIN_TRAN;

  procedure COMMIT_TRAN;

end;
```

- Tip #1 - Abstract the services into PL/SQL packages

  Although the Oracle Forms/Tuxedo interface makes PL/SQL versions of the Tuxedo client program elements available, they are generally too low-level for building Tuxedo clients (forms) directly. Abstract the higher-level services, and implement them in a PL/SQL package. Consider including a layer of "utility functions." The PL/SQL packages can reside in one or more libraries.

  Another benefit of this approach is that the utility functions can be reused by other bank applications, enabling quicker development times as well as supporting customer-specific processing standards.

- Tip #2 - Special considerations for `tpcall()`

  One of the bank utility functions is called `CALL_SERVICE()`in the BANKUTL Package Body. Note that `CALL_SERVICE()`calls `tpcall()`. The comment is helpful, but the

situation merits a closer look. The function is reproduced below so that you may refer to it in this discussion.

```
-- Note for CALL_SERVICE:
--
-- To be sure that we can catch a reallocation of fbfr by tpcall, we
-- don't use the passed-in fbfr.  Another problem is that we'd like
-- to raise an exception on failure, however we'd lose the pointer to
-- the reallocated fbfr, since the OUT var won't go back to the caller...
-- So instead, we return an error, and the simplest thing for the
-- caller to do is wrap CALL_SERVICE in a begin/end block, and raise the
-- TPM_FAILURE exception themselves.  This is gross, but typical of some
-- of the trickiness inherent in keeping two very different languages
-- (C and PL/SQL) in sync with each other.
--
function CALL_SERVICE (svcname in out varchar2,
                       fbfr    in out ora_ffi.pointertype,
                       buflen  in out pls_integer)
return pls_integer is
  fbfr1 ora_ffi.pointertype := fbfr;
  fbfr2 ora_ffi.pointertype := fbfr;
  flags pls_integer         := tuxdef.TPSIGRSTRT;
  retlen  pls_integer       := 0;
begin

  ret := atmi.tpcall (svcname, fbfr1, buflen, fbfr2, retlen, flags);

  --
  -- If the return length is non-zero, it means that reallocation
  -- occurred, and we have to set the buffer and length to the
  -- new address and size.
  --
  if retlen != 0 then
  fbfr   := fbfr2;
  buflen := retlen;
  end if;

  if ret = -1 then
    if atmi.gettperrno = TUXDEF.TPESVCFAIL then
      bankdef.errcat := 'SERVICE';
    else
      bankdef.errcat := 'TP';
    end if;
    bankdef.errtyp := 'TPCALL';
  end if;

  return (ret);
end;
```

There are two issues here. The first is that `atmi.tpcall()` may reallocate the fielded buffer, for example, to increase the size of the fielded buffer so as to contain the data from the reply. Distinct pointer variables (`fbfr, fbfr1,` and `fbfr2`) are used to preclude any confusion.

The second issue is how to handle an error returned by `atmi.tpcall()` and not lose the pointer to the fielded buffer (`fbfr`). This pointer is needed so that the calling program can free the fielded buffer if an error is detected. The solution is apparent from the comment and the code, nevertheless, it is instructive to see how the error is handled by the calling routine. The procedure, `OPEN()`, in the BANKSVCS PL/SQL package, is just such a calling routine. It is reproduced below.

```
-- Globals useful for all services
--
fbfr    ora_ffi.pointertype;  -- Fielded Buffer Pointer
buflen  pls_integer := 1024;  -- Fielded buffer length
ret     pls_integer;          -- Tuxedo return code
numbuf  varchar2(40);         -- Buffer for numeric conversions

procedure OPEN (lastname    in out varchar2,
                firstname   in out varchar2,
                midinitial  in out varchar2,
                address     in out varchar2,
```

```
                ssn         in out varchar2,
                phone       in out varchar2,
                initbalance in out number,
                accttype    in out varchar2,
                branchid    in out pls_integer,
                account_id  in out pls_integer,
                openbalance in out number,
                errmsg      in out varchar2) is
begin
  errmsg := null;
  numbuf := TO_CHAR(initbalance);
  fbfr   := bankutl.alloc_mem (FMLSTR, buflen);

  bankutl.set_value (fbfr, FNM_LAST_NAME,  0, lastname);
  bankutl.set_value (fbfr, FNM_FIRST_NAME, 0, firstname);
  bankutl.set_value (fbfr, FNM_MID_INIT,   0, midinitial);
  bankutl.set_value (fbfr, FNM_SSN,        0, ssn);
  bankutl.set_value (fbfr, FNM_ADDRESS,    0, address);
  bankutl.set_value (fbfr, FNM_PHONE,      0, phone);
  bankutl.set_value (fbfr, FNM_ACCT_TYPE,  0, accttype);
  bankutl.set_value (fbfr, FNM_BRANCH_ID,  0, branchid);
  bankutl.set_value (fbfr, FNM_SAMOUNT,    0, numbuf);
  bankutl.begin_tran;
  begin
    if bankutl.call_service (SVC_OPEN, fbfr, buflen) = -1 then
      raise bankdef.TPM_FAILURE;
    end if;
  end;
  bankutl.commit_tran;
  openbalance := bankutl.get_dollar  (fbfr, FNM_SBALANCE, 0);
  account_id  := bankutl.get_number  (fbfr, FNM_ACCOUNT_ID, 0);
  bankutl.free_mem (fbfr);
exception
  when bankdef.ALLOCATION_FAILURE then
    bankutl.compose_error (fbfr, errmsg);
  when bankdef.TPM_FAILURE then
    bankutl.compose_error (fbfr, errmsg);
    bankutl.free_mem (fbfr);
    ret := atmi.tpabort(0);
end;
```

Note the begin/end block in the middle of the procedure to raise the exception. The exception handler further below frees the fielded buffer and aborts the transaction by directly using an ATMI call, `atmi.tpabort()`.

Although `atmi.tpabort()` was called directly, it could just as easily have been wrapped by a bank utility function, similar to `bankutl.begin_tran()` or `bankutl.commit_tran()`; not doing so technically violates the layered approach recommended earlier.

3.3.1.2 bankapp Client PL/SQL Form

There is a non-trivial amount of code, primarily to support the various triggers in the bankapp client PL/SQL form, that is not in the bankapp client PL/SQL library. To explore the source code in the bankapp client PL/SQL form, use the Object Navigator in Form Builder to open the bankapp Form module binary file, BANKAPP.FMB. Expand the BANKAPP node to reveal the form's object hierarchy. The first objects of interest are the Triggers. Expand "Triggers" to view the three triggers that were customized for the bankapp client form. To see the PL/SQL code behind each trigger, double-click on the trigger icon. The trigger's code appears in the PL/SQL Editor.

Read the comments in each of the triggers. Note that the POST-FORM trigger calls an interface layer function, `d2tx.shutdown()`, directly. The ON-LOGON trigger contains nothing more that a null statement. This is to prevent Oracle Forms from executing its default logon processing, which wouldn't make any sense in the context of a TP monitor.

- Tip #3 - Consider the Forms built-in triggers

When you build an application, Oracle Forms' built-in triggers must be taken into account, particularly when the form (client) will be not be interfacing directly with an Oracle data source, as is the case with Tuxedo. In many cases, the default processing will have to be suppressed, as was the case with the ON-LOGON transactional trigger discussed above. But in many instances, these triggers also provide a convenient location to place code that will interact appropriately with the TP monitor.

Continuing with the tour of the bankapp client form, the next objects of interest are the Data Blocks. Expand the "Data Blocks" node to see the three data blocks in this form. Data Blocks provide a mechanism for grouping related items into a functional unit for storing, displaying, and manipulating records. These data blocks correspond to the three screens the form displays at one time or another.

Of particular interest is the trigger code associated with each button. To illustrate the point, the code for the WHEN-BUTTON-PRESSED trigger under the item called "VERB," under the data block called "ACTIONS," is listed below. "VERB" is a generic reference for the OK button that appears at the bottom of the "ACTIONS" screen. Depending on exactly what the action is, the trigger code calls the appropriate bank service routine, for example, `banksvcs.inquiry()`. This is another illustration of the form layer relying on routines from the bankapp client PL/SQL library.

```
-- Copyright (C) Oracle Corporation 1996, 1998.
-- All Rights Reserved, Worldwide.

declare
  balance1 number;
  balance2 number;
  account1 pls_integer  := :actions.account1;
  account2 pls_integer  := :actions.account2;
  amount   number       := :actions.amount;
  action   varchar2(20) := :bank_svcs.services;
  errmsg   varchar2(250);
  discard  number;
  item1    varchar2(30) := null;
  item2    varchar2(30) := null;

begin
  --
  -- Call the appropriate service for the current action
  -- (We also use this block to display initial balance after
  --  creation so if the action is OPEN then we just go back to
  --  the main block)
  --
  if (action = 'OPEN_ACCT') then
    go_block ('bank_svcs');
    return;
  elsif (action = 'INQUIRY') then
    banksvcs.inquiry (account1, balance1, errmsg);
    :actions.balance1 := balance1;
  elsif (action = 'DEPOSIT') then
    banksvcs.deposit (account1, amount, balance1, errmsg);
    :actions.balance1 := balance1;
  elsif (action = 'WITHDRAW') then
    banksvcs.withdraw (account1, amount, balance1, errmsg);
    :actions.balance1 := balance1;
  elsif (action = 'CLOSE_ACCT') then
    banksvcs.close (account1, balance1, errmsg);
    :actions.balance1 := balance1;
  elsif (action = 'TRANSFER') then
    banksvcs.transfer (account1, account2, amount, balance1, balance2, errmsg);
    :actions.balance1 := balance1;
    :actions.balance2 := balance2;
    item1 := 'actions.bal2_label';
    item2 := 'actions.balance2';
    null;
```

```
  else
    errmsg := 'INTERNAL ERROR: Unknown transaction type';
  end if;

  -- If the service returned an error, display it
  --
  if (errmsg is not null) then
    hideitem ('actions.bal1_label');
    hideitem ('actions.balance1');
    hideitem (item1);
    hideitem (item2);
    synchronize;
    set_alert_property ('ERRORMSG', ALERT_MESSAGE_TEXT, errmsg);
    discard := show_alert ('ERRORMSG');
  else
    showitem ('actions.bal1_label');
    showitem ('actions.balance1');
    showitem (item1);
    showitem (item2);
  end if;
end;
```

In the Object Navigator, move down to the node called "Canvases." Expand this node, then double-click a canvas icon to see how a screen will appear when the form is running.

Finally, expand the Program Units node to see the PL/SQL functions and procedures that are associated with this form. Since these routines are really only specific to this particular form, they are found here rather than in the Oracle Forms bankapp client PL/SQL library.

4. Appendix

4.1 What's New in this Release?

4.1.1 Bug Fixes

This section highlights the improvements that are featured in this release.

- Bug #1201146
  Minor improvements and updates were made to this white paper.

4.1.2 Current Limitations

- Asynchronous ATMI client functions are not supported in this release.

4.2 Frequently Asked Questions

This section answers some questions related to the Oracle Forms/Tuxedo Interface.

4.2.1 General

· *Isn't there some other interface between Oracle and Tuxedo?*

Yes, there is, but it's a little different from this one. That interface is between an Oracle *database* and Tuxedo using the standard XA protocol. The Oracle database fulfills the role of the data management service on the resource server (third tier), while the Oracle Forms/Tuxedo Interface enables the development of Tuxedo clients for the desktop (the first tier). These interfaces are complementary.

There's even a demo of this that also uses bankapp. It shows an Oracle database (Oracle7) acting as the database for the bankapp, rather than using the internal data structures that are shipped with bankapp. This demo uses the data dependent routing feature of the Tuxedo system. For more information about this database interface or its demo, see the draft white paper *INTEGRATING THE TUXEDO SYSTEM WITH ORACLE 7 RDBMS*, dated 17 April 1995. It should be available from BEA Systems.

4.2.2 Marketing

· *Are other interfaces available or planned for more recent releases of Tuxedo?*

This interface is with Tuxedo System Release 6.4. Interfaces supporting more recent releases of Tuxedo can be expected if the market demands them.

· *Will there be an interface of FML32 available at some point?*

Yes, if there is enough demand from the marketplace to justify the effort.

· *What TP monitor interfaces are available or planned for Oracle Forms?*

Oracle Corporation developed a prototype with a similar interface with Digital Equipment Corporation's ACMS Desktop. NCR Corporation (http://www.ncr.com/) has developed an interface between Oracle Forms and their TP monitor, TOP END, which is available from NCR.

4.3 Additional Resources

There are many other resources available to aid in the understanding of this interface, as well as its constituent and enabling technologies.

4.3.1 Oracle Forms

The following additional resources are available for Oracle Forms:

4.3.1.1 On-line Documentation

- There is a wealth of knowledge in the Oracle Forms on-line documentation. Of particular interest would be the sections that discuss the PL/SQL interface to foreign functions and transactional triggers. These can both be found in the Form Builder online help index (Foreign functions, Transactional Triggers).
- The Procedure Builder online help has an entire node devoted to calling functions in dynamic libraries under the heading "Building and Running a Program Unit," as well as a detailed description of the ORA_FFI (foreign function interface) built-in package in the PL/SQL Reference.

4.3.1.2 White Papers

- Additional white papers are available from the Oracle Technology Network (http://technet.oracle.com/). Contact your Oracle Corporation Sales Representative or Consultant for more information about these resources.

4.3.1.3 Books

- Feuerstein, Steven, with Bill Pribyl. *ORACLE PL/SQL Programming, 2nd Edition*. Sebastopol, CA: O'Reilly & Associates, Inc., October 1997. ISBN: 1565923359. A very rich tome covering just about everything anyone would want to know about PL/SQL.

4.3.2 Tuxedo and TP Monitors

The following additional resources are available for Tuxedo:

4.3.2.1 Documentation Set

- Of course, the *BEA TUXEDO System 6 Installation Guide* is essential to getting started. Pay particular attention to the sections devoted to configuring the operating system, and the data sheet for the operating system under which Tuxedo will run; it is almost guaranteed that at least some of the kernel-tunable parameters will have to be adjusted to get bankapp to work correctly.

The remainder of Tuxedo's product documentation is installed with the product as HTML documents. The following Tuxedo documents are the most relevant to this interface.

- Refer to the */Workstation Guide* for information about how to bring up Tuxedo's bankapp client on the client machine, and for information on how to design and write Tuxedo clients.
- The *Application Developer's Guide* contains information about how to develop a Tuxedo application, using the bankapp as an example.

- Everything you wanted to know about Tuxedo's Form Manipulation Language is in the *FML Programmer's Guide*.
- For more programming information, refer to the *Programmer's Guide*, especially Chapter 2, "Writing Client Programs."
- The *Reference Manual: Section 3C* contains detailed descriptions for the ATMI C functions and the *Reference Manual: Section 3FML* contains detailed descriptions for the FML C functions.

4.3.2.2 White Papers

- *Programming a Distributed Application* is a good description of the four communication techniques available to programmers using Tuxedo to write distributed applications. This is available from BEA Systems.
- *Exploring TUXEDO Using the bankapp Demo Program*, written by C. Cash Perkins, and dated 12/7/95, is a good resource for understanding how to get bankapp to run. This is also available from BEA Systems.

4.3.2.3 Books

- Grey, Jim and Reuter, Andreas. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, 1993. ISBN 1-55860-190-2. This book is widely considered to be *the* authoritative reference book for TP systems.
- Hall, Carl L. *Building Client/Server Applications using Tuxedo*. John Wiley & Sons, Inc., 1993. ISBN 0-471-12958-5.
- Primatesta, Fulvio. *Tuxedo: An Open Approach to OLTP*. Prentice Hall, 1995. ISBN 0-13-101833-7

4.3.2.4 Web Pages

- The URL for Tuxedo information is http://www.beasys.com.

Using Oracle Forms with the Tuxedo TP Monitor
April 2000

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
650.506.7000
Fax 650.506.7200