Oracle® Tuxedo System and Applications Monitor Plus

Programming Guide

Release 22c
F93048-01
July 2026

Oracle Tuxedo System and Applications Monitor Plus Programming Guide, Release 22c

F93048-01

Primary Authors: Preeti Gandhe, Priya Pathak

Contributing Authors: Tulika Das

Contributors: Maggie Li

Contents

List of Tables

1

Oracle Tuxedo JMX Interface

This chapter contains the following sections:

1.1 Overview

The JMXAgent embedded in tlisten supplies a list of JMX MBeans. Using the functions exported by those MBeans, users can monitor and manage Oracle Tuxedo application by JMX invocation. Following is an example of shutting down a Tuxedo Server by JMX Java client.

Shutting Down a Tuxedo Server by JMX Java Client

```
// Client.java
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import javax.management.MBeanServerConnection;
import javax.management.ObjectInstance;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;

import oracle.tuxedo.jmx.tux.exception.JMXPropertiesException;
import oracle.tuxedo.jmx.tux.utility.Encryption;

public class Client {

    public static void main(String[] args) throws IOException,
JMXPropertiesException, Exception {
        String host = "XXXXXX.oracle.com";
        String port = "26999";
        String username = "oracle";

        // Tuxedo password and application password must be encrypted.
        String password = Encryption.getInstance().encrypt("password");

        // Tuxedo application password.
        String appPassword = Encryption.getInstance().encrypt("apppassword");

        String[] credentials = new String[] { username, password,
appPassword };
```

```
        Map<String, Object> env = new HashMap<>();
        env.put("jmx.remote.credentials", credentials);

        // Create an RMI connector client and connect it to the RMI connector
server.
        JMXServiceURL url =
                new JMXServiceURL(
                        String.format(
                                "service:jmx:rmi://%s:%s/jndi/rmi://%s:%s/
server",
                                host,
                                port,
                                host,
                                port));

        JMXConnector jmxc = JMXConnectorFactory.connect(url, env);
        MBeanServerConnection mbsc = jmxc.getMBeanServerConnection(null);

        // Get AdminBean by the MBean name.
        Set<?> mbeans = mbsc.queryMBeans(new
ObjectName("DefaultJMXDomain:type=adminBean"), null);
        ObjectInstance objectInstance = (ObjectInstance)
mbeans.iterator().next();
        ObjectName objectName = objectInstance.getObjectName();

        // The parameter value list for shutdownServer method.
        Object[] params = { "/testarea/oracle/test/tuxconfig", "APPGRP", 102,
0, null };

        // The parameter type list for shutdownServer method.
        String[] signature = {
            String.class.getName(),
            String.class.getName(),
            Integer.class.getName(),
            Integer.class.getName(),
            String.class.getName()
        };

        Object result = mbsc.invoke(objectName, "shutdownServer", params,
signature);
        System.out.print(result);
    }
}
```

Run the following command to compile the java file:

```
javac -classpath $TUXDIR/jmx/tmjmx_exceptions.jar:$TUXDIR/jmx/tmjmx_tux.jar
Client.java
```

A Client.class is generated. Use the following command to run this client:

```
java -classpath $TUXDIR/jmx/tmjmx_exceptions.jar:$TUXDIR/jmx/tmjmx_tux.jar:.
Client
```

The output result is as follows:

```
Shutting down server processes ...
```

```
Server Id = 102 Group Id = APPGRP Machine = SITE1: shutdown succeeded 1 process
stopped.
```

For more information, see MBeans and JMX Operations.

1.2 Configurations

Following configurations must be done before invoking the JMX client:

- Start the tlisten and Tuxedo domain. For more information, refer to “Starting the tlisten Process”, “Configuring the UBBCONFIG File", and "Loading UBBCONFIG and Booting up Oracle Tuxedo" sections in Enterprise Manager for Oracle Tuxedo Getting Started Guide.
- For the Tuxedo domain which security is not “NONE”, add the Tuxedo user as tpsysadm. For example:

  ```
  tpusradd -g group1 -c tpsysadm user1
  ```

- Add $TUXDIR/udataobj/jmx/tmjmx_exceptions.jar to the client’s classpath when running the client to invoke the JMX Agent in tlisten.

1.3 Creating a JMX Connection

Use the following JMX Service URL to create a JMX connection:

```
service:jmx:rmi://rmihost:rmiport/jndi/rmi://rmihost:rmiport/server
```

The rmihost and rmiport are the host and port configured in tlisten by the -j option. After configuring this option, a JMX Agent runs in a JVM started in the tlisten process by JNI technology.

The "jmx.remote.credentials" must be a String array which is parsed in the following order:

Order	Tuxedo Authentication and Authorization Data	Comments
0	User Name	Optional. Required when the Tuxedo security or JMX security is enabled; otherwise leave it blank. If the Tuxedo security is enabled, it is the Tuxedo username; If JMX security is enabled, it is the JMX user name added by jmxaaacfg.
1	Password	Optional. Required if the Tuxedo security or JMX security is enabled; otherwise leave it blank. If the password is not null, it must be encrypted using the oracle.tuxedo.jmx.tux.utility.Encryption.encrypt() method in $TUXDIR/jmx/tmjmx_tux.jar
2	Application Password	Optional. Required if the Tuxedo security is enabled; otherwise leave it blank. If the application password is not null, it must be encrypted using the oracle.tuxedo.jmx.tux.utility.Encryption.encrypt() method in $TUXDIR/jmx/tmjmx_tux.jar
3	DOMAINID	Optional. Required in the case of multiple domains within one tlisten.
4	IPCKEY	Optional. Required in the case of multiple domains within one tlisten.
5	TUXCONFIG	Optional. Required after the tlisten restarts.

Order	Tuxedo Authentication and Authorization Data	Comments
6	NONTUXAUTH	Optional. Add "NONTUXAUTH" in the list of "jmx.remote.credentials" , the Tuxedo authentication is not taken place when connecting JMX server. In this case, the invocation of the JMX operations which requires Tuxedo authentication will fail.

Following is an example of creating JMX connector.

Creating JMX Connector

```
String[] credentials = new String[] { username, password, appPassword };
Map<String, Object> env = new HashMap<String, Object>();
env.put("jmx.remote.credentials", credentials);

// Create an RMI connector client and
// connect it to the RMI connector server;
JMXServiceURL url = new JMXServiceURL(String.format(
"service:jmx:rmi://%s:%s/jndi/rmi://%s:%s/server", host, port,
host, port));

JMXConnector jmxc = JMXConnectorFactory.connect(url, env);
```

2 Extended Call Path Monitoring

This chapter contains the following sections:

2.1 Overview

Extended call path monitoring enables you to set customized call path segments in Tuxedo service calls using the API `tsambegin()` and `tsamend()`.

To enable extended call path monitoring, in the Oracle TSAM Plus console, select **Enable Extended Monitoring** in the call path tab of the Policy page. For more information, see Call Path Tab in Using Oracle TSAM Plus Console

2.2 Using tsambegin()/tsamend() in Oracle Tuxedo Service

The following is an example that defines `tsambegin()`and `tsamend()` in the file `toupper`:

Defining tsambegin() and tsamend()

```
#include <tsam_ext.h>
void
TOUPPER(TPSVCINFO *rqst)
{
...
long seq;
int db_rtn;
char * begin_props[] = { "sql=update..."};
char * end_props[1];

seq = tsambegin("DB", "Update", 1, begin_props, 0L);
userlog("seq=%ld", seq);


if(seq >= 0){
char str_dbrtn[100];

sprintf(str_dbrtn, "Database Return=%d", db_rtn);
end_props[0] = str_dbrtn;
tsamend(seq, sizeof(end_props)/sizeof(end_props[0]), end_props, 0L);
}
....
}
```

Run the following command to build the file `toupper.c`:

```
buildserver -s TOUPPER -o toupper -f toupper.c -f ${TUXDIR}/lib/tsam_ext.o
```

For more information about `tsambegin()`and `tsamend()`, see Call Path Monitoring APIs in Oracle TSAM Plus Reference Guide.

3

Developing Oracle TSAM Plus Agent Custom Plug-in

This chapter contains the following sections:

3.1 Overview

The Oracle TSAM Plus Agent includes three major layered modules:

- Oracle TSAM Plus framework
  The Oracle TSAM Plus framework is responsible for Tuxedo system data collection. The collection behavior is controlled by the monitoring types and policies. The gathered metrics are passed to the plug-in using an open interface.

  Plug-in data receiver

  The Oracle TSAM Plus Agent ships with a default plug-in. The default plug-in sends metrics to shared memory pool created by the Local Monitor Server (LMS).
- LMS (local monitor server)
  The LMS synchronizes data with the Oracle TSAM Plus Manager.

The Oracle TSAM Plus Agent and Oracle TSAM Plus Manager provide a complete solution for data collection, aggregation, storage and presentation. To support various requirements for monitoring data usage, the Oracle TSAM Plus Agent plug-interface is based on an open architecture so that you can write customized plug-ins to interpret the performance metrics data. The custom plug-ins can work with the Oracle TSAM Plus Agent default plug-in or independently. The custom plug-ins are typically used for:

- Integration with third party management software
- Developing in-house application monitoring suites
- Audit-based application data

3.2 Oracle TSAM Plus Agent Data Collection Framework

The Oracle TSAM Plus Agent framework collects the performance metrics when Oracle TSAM Plus is enabled. The framework covers the major performance sensitive areas in Tuxedo applications, that is call path stages, services, transactions and system servers. Oracle TSAM Plus Agent uses Oracle Tuxedo FML32 typed buffers to contain the metrics collected so that each metric is defined as a built-in FML32 field. The monitoring points depend on the monitoring types and only apply to Oracle Tuxedo ATMI applications. The following table lists the call path monitoring points.

Table 3-1 Call Path Monitoring Points

Stage	Supported Tuxedo Process Types
Before request message sent to IPC queue	Native Client, Application Server, GWTDOMAN, BRIDGE, JSH/WSH, and GWWS
After request message got from IPC queue	Application Server, GWTDOMAIN and GWWS
Before reply message sent to IPC queue	Application Server, GWTDOMAIN, BRIDGE and GWWS
After reply message got from IPC queue	Native Client,Application Server, GWTDOMAIN
Before request message sent to network	GWTDOMAIN, JSH/WSH, and GWWS
After request message got from network	GWTDOMAIN and BRIDGE
Before reply message sent to network	GWTDOMAIN and BRIDGE
After reply message got from network	GWTDOMAIN and BRIDGE

The following table lists the service monitoring points.

Table 3-2 Service Monitoring Points

Stage	Supported Tuxedo Process Types
After request from IPC queue	Application Server, GWTDOMAIN and GWWS
Before reply message sent to IPC queue	Application Server, GWTDOMAIN and GWWS

The following table lists the system server monitoring points.

Stage	Supported Tuxedo Process Types
Main Loop	GWTDOMAIN, BRIDGE and GWWS. The metrics are collected internally and this point is to pass the data to plug-in

> **Note**
>
> The metrics are collected internally, and the Main Loop passes the data to plug-in

Table 3-3 Transaction Monitoring Points

Stage	Supported Tuxedo Process Types
After the transaction routine executed	Native Client, Application Server, TMS, GWTDOMAIN,WSH, JSH, TMQFORWARD

3.3 Creating an Oracle TSAM Plus Agent Custom Plug-in

This section contains the following topics:

- Overview
- Developing a Oracle TSAM Plus Agent Plug-in
- Oracle TSAM Plus Agent Plug-in Interface
- Oracle TSAM Plus Agent Plug-in Implementation

- Oracle TSAM Plus Agent Plug-in Development/Deployment Notes

3.3.1 Overview

Oracle Tuxedo has a built-in plug-in framework that facilitates additional functionality. For example, the Oracle Tuxedo security mechanism is constructed on the plug-in framework. Oracle Tuxedo defines an *interface set* as a contract between a service provider and end user. The term “service” here is used as a general term; not an Oracle Tuxedo ATMI service. Oracle TSAM Plus Agent also use the Oracle Tuxedo plug-in framework to attach different data receivers.

- Oracle Tuxedo Plug-in Framework Concepts

3.3.1.1 Oracle Tuxedo Plug-in Framework Concepts

The following section highlights Oracle Tuxedo plug-in framework key concepts:

- Interface
- Implementation
- Plug-in Register/Un-register/Modifications

3.3.1.1.1 Interface

An Interface is the contract format between the plug-in implementation and the plug-in caller. An interface requires the following attributes:

- **Interface ID**
  The interface ID is the name of the interface that is uniquely identified in the Oracle Tuxedo plug-in framework and uses the following format:

  ```
  <interface id> ::= <component name>[/<sub-component/name>]/<interface name>
  ```

  The Oracle TSAM Plus Agent plug-in uses the following format:

  ```
  engine/performance/monitoring
  ```

- **Version**
  An interface has two versions, the major version number and minor version number.
- **Data Structure and Function Declaration**
  The data structure defines the concrete information conveyed between plug-in caller and implementation. The function declaration defines the routines must be implemented by plug-in.

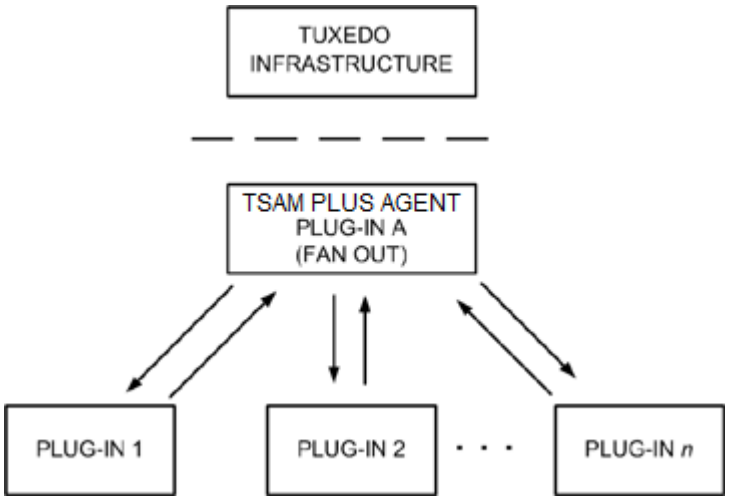
  An interface has two versions, the major version number and minor version number.
- **Data Structure and Function Declaration**
  The data structure defines the concrete information conveyed between plug-in caller and implementation. The function declaration defines the routines must be implemented by plug-in.

3.3.1.1.2 Implementation

A plug-in is a dynamic library written in C code. The library implements the methods specified by the interface. The Oracle Tuxedo plug-in framework supports multiple implementations (interceptors) for one interface.

Oracle Tuxedo supports two types of interceptors: Fan-out interceptors and Stack interceptors. The Oracle TSAM Plus Agent uses the Fan-out interceptors. The following figure displays the Oracle TSAM Plus Agent plug-in architecture.

Figure 3-1 Oracle TSAM Plus Agent Plug-in Architecture



When the Oracle Tuxedo infrastructure invokes plug-in A method X, plug-in A invokes method X of the intercepting plug-ins in the order specified by the `InterceptionSeq` attribute as follows:

- Plug-in 1 method is invoked
- Plug-in 1 method X is returned
- Plug-in 2 method X is invoked
- Plug-in 2 method X is returned
- Plug-in *n* method X is invoked
- Plug-in *n* method X of is returned

All plug-ins involved in the interceptor implement the same interface. Multiple occurrences of the same plug-in are not allowed in an interception sequence.

Oracle TSAM Plus Agent provides the Fan-out plug-in which allows you to write/create an interceptor plug-in.

3.3.1.1.3 Plug-in Register/Un-register/Modifications

Once the plug-in written it must be registered in the Oracle Tuxedo registry so that the functional components will locate the plug-in and invoke the appropriate methods. Oracle Tuxedo provides three commands specifically for plug-in use:

- `epifreg`: registers a plug-in
- `epifunreg`: un-registers a plug-in
- `epifregedt`: edits a plug-in

3.3.2 Developing a Oracle TSAM Plus Agent Plug-in

Oracle TSAM Plus Agent plug-in invocation begins at the monitoring points. The Oracle TSAM Plus Agent collects and computes the metrics, and composes the arguments passed to the plug-in. The Oracle TSAM Plus Agent Fan-out plug-in invokes the interceptor plug-in according to the registration sequence.

A simple Oracle TSAM Plus custom plug-in development example is provided as a guideline. The system environment is Solaris on Sparc. The functionality is basic and just prints out the metrics buffers. This plug-in works together with the Oracle TSAM Plus Agent default plug-in.

3.3.2.1 Create Plug-in Source Code

The following listing displays an example of the Oracle TSAM Plus plug-in `customplugin.c`.

The following code snippet is an example of `customplugin.c` Plug-in Source Code:

Oracle TSAM Plus Agent `customplugin.c` Plug-in Source Code Example

```
#include <e_pif.h>
#include <tpadm.h>
#include <fml32.h>
#include <e_perf_mon.h>

static TM32I _TMDLLENTRY print_app(
       perf_mon_1 *,
       FBFR32 **,
       MONITORCTL *,
       TM32U);

static TM32I _TMDLLENTRY print_svc(
       perf_mon_1 *,
       FBFR32 **,
       MONITORCTL *,
       TM32U);

static TM32I _TMDLLENTRY print_sys(
       perf_mon_1 *,
       FBFR32 **,
       MONITORCTL *,
       TM32U);

static TM32I _TMDLLENTRY print_tran(
       perf_mon_1 *,
       FBFR32 **,
       MONITORCTL *,
       TM32U);
```

```
static TM32I _TMDLLENTRY plugin_destroy (
       _TCADEF,
       const struct _e_pif_instance_handles *,
       TM32U);

static TM32I _TMDLLENTRY plugin_copy (_TCADEF,
       void *,
       const struct _e_pif_interception_data *,
       struct _e_pif_instance_handles *,
       TM32U);

static const perf_mon_1 Vtblperfapp_1 = {
       print_app,
       print_svc,
       print_sys,
       print_tran,
};

static const _e_pif_plugin_info perf_mon_1_info = {
       { 1, 0 }, /* interface major version */
       { 1, 0 }, /* implementation */
       "abc/tuxedo/tsam", /* implementation id */
       ED_PERF_MON_INTF_ID, /* interface id */
       4, /* virtual table size */
       "ABC, Inc.", /* vendor */
       "Custom Plug-in for Oracle TSAM", /* product name */
       "1.0", /* vendor version */
       EF_PIF_SINGLETON, /* m_flags */
       plugin_destroy,
       plugin_copy
};

int _TMDLLENTRY
plugin_entry(_TCADEF, const char *pIId,
       const char *pImplId,
       const struct _e_pif_iversion *version,
       const struct _e_pif_data *pData,
       const struct _e_pif_interception_data *pInterceptionData,
       struct _e_pif_instance_handles *pI,
       TM32U flags)
{
       const char * const * regData = pData->regdata;
        char *logfile = NULL;

       pI->pVtbl = (void *) &Vtblperfapp_1;
       pI->pPluginInfo = (_e_pif_plugin_info *) &perf_mon_1_info;
       pI->pPrivData = NULL;
return (EE_SUCCESS);
}


static TM32I _TMDLLENTRY
plugin_destroy (_TCADEF, const struct _e_pif_instance_handles *pIhandles,
       TM32U flags)
{
```

```
        return(EE_SUCCESS);
}


static TM32I _TMDLLENTRY
plugin_copy (_TCADEF, void *iP,
        const struct _e_pif_interception_data *pInterceptionData,
        struct _e_pif_instance_handles *pIhandles,
        TM32U flags)
{
        return(EE_SUCCESS);
}
static TM32I _TMDLLENTRY print_app(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL *
monctl, TM32U flags)
{
        Fprint32(*buf);
        return(0);
}

static TM32I _TMDLLENTRY print_svc(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL *
monctl, TM32U flags)
{
        Fprint32(*buf);
        return(0);
}

static TM32I _TMDLLENTRY print_sys(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL *
monctl, TM32U flags)
{
        Fprint32(*buf);
        return(0);
}
static TM32I _TMDLLENTRY print_tran(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL
* monctl, TM32U flags)
{
        Fprint32(*buf);
        return(0);
}
```

3.3.2.2 Build the Plug-in

```
cc -c customplugin.c -I$TUXDIR/include

cc -G -KPIC -o customplugin.so -L$TUXDIR/lib -lfml customplugin.o
```

3.3.2.3 Register the Plug-in

To register the plug-in, do the following steps:

1. Shutdown your Oracle Tuxedo application by “tmshutdown”
2. Compose a shell script named “`reg.sh`”
3. Run the script `sh ./reg.sh`
4. Boot your Oracle Tuxedo applications by "tmboot"

The following code snippet is an example of the reg.sh shell script

reg.sh Shell Script

```
#!/bin/sh
epifreg -r -p abc/tuxedo/tsam -i engine/performance/monitoring \
-o SYSTEM -v 1.0 \
-f $APPDIR/customplugin.so -e plugin_entry
epifregedt -s -k "SYSTEM/impl/bea/performance/monfan" \
-a InterceptionSeq=bea/performance/monshm \
-a InterceptionSeq=abc/tuxedo/tsam \
```

3.3.2.4 Enable Oracle TSAM Plus Monitoring

Enable Oracle TSAM Plus Monitoring by defining the proper monitoring policy through Oracle TSAM Plus console

For more information, see Oracle TSAM Plus Users Guide.

3.3.2.5 Run a Call and Check the Standard Output

You will find the metrics collected printed out.

Metrics Print Out Example

```
TA_MONDEPTH 1
TA_MONSTATUS 1
TA_MONPROCTYPE 2
TA_PID 2459
TA_SRVID 10
TA_MONLOGTIMESEC 1259292914
TA_MONLOGTIMEUSEC 26411
TA_MONFIELDSMAP1 -1
TA_MONFIELDSMAP2 -1
TA_MONMSGSIZE 24
TA_MONMSGQUEUED 0
TA_MONLASTTIMESEC 1259292914
TA_MONLASTTIMEUSEC 26411
TA_MONSTARTTIMESEC 1259292914
TA_MONSTARTTIMEUSEC 10500
TA_MONELAPSETIME 15
TA_DOMAINID dom2:bjsol16:66536
TA_GROUPNAME ATMIGRP1
TA_LMID L1
TA_MONTYPE APP
TA_MONCORRID dom2:bjsol16:66536 L1 tuxclient 2478 1 1 1259292909
TA_MONMSGTYPE ARQ
TA_MONSTAGE Q2ME
TA_MONSVCNAME I_TOUPPER
TA_MONHOSTSVC I_TOUPPER
TA_MONSVCSEQ INITIATOR-I_TOUPPER-11659-0
TA_MONPSVCSEQ INITIATOR
TA_MONQID 1879048194-00010.00010
TA_MONPROCNAME tux_atmi_svr

TA_MONDEPTH 1
TA_MONSTATUS 1
```

```
TA_MONPROCTYPE 2
TA_PID 2459
TA_SRVID 10
TA_MONLOGTIMESEC 1259292914
TA_MONLOGTIMEUSEC 29368
TA_MONFIELDSMAP1 -1
TA_MONFIELDSMAP2 -1
TA_MONMSGSIZE 100
TA_MONLASTTIMESEC 1259292914
TA_MONLASTTIMEUSEC 29368
TA_MONSTARTTIMESEC 1259292914
TA_MONSTARTTIMEUSEC 10500
TA_MONERRNO 0
TA_MONURCODE 1
TA_MONELAPSETIME 18
TA_DOMAINID dom2:bjsol16:66536
TA_GROUPNAME ATMIGRP1
TA_LMID L1
TA_MONTYPE APP
TA_MONCORRID dom2:bjsol16:66536 L1 tuxclient 2478 1 1 1259292909
TA_MONMSGTYPE ARP
TA_MONSTAGE ME2Q
TA_MONSVCNAME I_TOUPPER
TA_MONHOSTSVC I_TOUPPER
TA_MONSVCSEQ INITIATOR-I_TOUPPER-11659-0
TA_MONPSVCSEQ INITIATOR
TA_MONPROCNAME tux_atmi_svr
```

3.3.3 Oracle TSAM Plus Agent Plug-in Interface

All Oracle TSAM Plus Plug-in interface contents are defined in the `$TUXDIR/include/e_perf_mon.h` file. When you build an Oracle TSAM Plus Plug-in, this file must be included in your plug-in source code.

The `$TUXDIR/include/e_perf_mon.h` file definitions are as follows:

- Version and Interface Identifier
- Function Table
- Other Help Header Files

3.3.3.1 Version and Interface Identifier

The following listing provides a version and identifier example.

Version and Interface Identifier

```
#define ED_PERF_MON_MAJOR_VERSION 1
#define ED_PERF_MON_MINOR_VERSION 0
/* Interfaces defined in this module */
#define ED_PERF_MON_INTF_ID "engine/performance/monitoring"
Value Definitions and Data Structure
```

The following listing shows the Oracle TSAM Plus framework and plug-in core data structure.

Core Data Structure

```
typedef struct {
unsigned char fieldsmap[MAXMAPSIZE];
char monitoring_policy[MAXPOLICYLEN]; /* monitor policy */
char corr_id[MAXCORRIDLEN]; /* plug-in supplied correlation ID */
int ulen;
void * udata;
long mon_flag;
} MONITORCTL;
```

The following table lists the MONITORCTL members.

Table 3-4 MONITORCTL Members

Members	Description
monitoring_policy	Internal use only
corr_id	It is used to bring the correlation ID from plug-in to TSAM Plus framework
ulen	The data length of the application buffer.
udata	The application buffer. It is a typed buffer and only available for call path monitoring and service monitoring. tptypes(5) can be used to check the type and subtype.
mon_flag	The flag set both by TSAM Plus framework and plug-in to indicate the requirement and changes.

The following table lists the MONITORCTL array size definitions.

Table 3-5 MONITORCTL Array Size Definitions

Array	Size Description
/* Size of fieldsmap*/	#define MAXMAPSIZE 128
/* Size of monitoring_policy */	#define MAXPOLICYLEN 128
/* Size of corr_id*/	#define MAXCORRIDLEN 256

The following table lists the mon_flag Values.

Table 3-6 mon_flag Values

Members	Description
#define PI_CORRID_REQUIRED 0x00000001	PI_CORRID_REQUIRED is set by TSAM Plus framework when a call path monitoring is started. It means the plug-in must supply a correlation ID to the framework by the corr_id member of MONITORCTL.

3.3.3.2 Function Table

The following listing defines the plug-in implementation method function table:

Plug-in Implementation Method Function Table

```
typedef struct perf_mon_1_Vtbl {
       TM32I (_TMDLLENTRY *_ec_perf_mon_app) _((
```

```
                        struct perf_mon_1_Vtbl * ip,
                        FBFR32 **buf,
                        MONITORCTL *mon_ctl,
                        TM32U flags
        ));
        TM32I (_TMDLLENTRY *_ec_perf_mon_svc) _((
                        struct perf_mon_1_Vtbl * ip,
                        FBFR32 **buf,
                        MONITORCTL *mon_ctl,
                        TM32U flags
        ));
        TM32I (_TMDLLENTRY *_ec_perf_mon_sys) _((
                        struct perf_mon_1_Vtbl * ip,
                        FBFR32 **buf,
                        MONITORCTL *mon_ctl,
                        TM32U flags
        ));
        TM32I (_TMDLLENTRY *_ec_perf_mon_tran) _((
                        struct perf_mon_1_Vtbl * ip,
                        FBFR32 **buf,
                        MONITORCTL *mon_ctl,
                        TM32U flags
        ));
} perf_mon_1, *perf_mon_1_ptr;
```

Each method corresponds to a monitoring type. "**_ec_perf_mon_app**" is for call path monitoring, "**_ec_perf_mon_svc"** is for service monitoring, "**_ec_perf_mon_sys"** is for system server monitoring and "**_ec_perf_mon_tran"** is for transaction monitoring. Each method will be invoked at the corresponding monitoring type's monitoring points. The method arguments are:

- `struct perf_mon_1_Vtbl *ip`:the virtual table pointer e.

  > **Note**
  >
  > Not required for custom plug-ins.

- `FBFR32 **buf`:the address of the metrics buffer in FML32 type.
- `MONITORCTL *mon_ctl`: the control structure.
- `TM32U flags`: the bit flag in a 32-byte, unsigned integer.

3.3.3.3 Other Help Header Files

- `$TUXDIR/include/e_pif.h`
  The Oracle Tuxedo general plug-in definition file. It must be included in the plug-in source code.
- `$TUXDIR/include/tpadm.h`
  It is the Oracle Tuxedo built-in FML32 fields definition files. All performance metrics are defined as FML32 fields and some performance metrics are defined in this file
- `$TUXDIR/include/monflds.h`
  `The TSAM built-in FML32 fields definition files. All performance metrics are defined in this file beside the tpadm.h`

- `$TUXDIR/include/fml32.h`
  The metrics collected are stored in an Oracle Tuxedo FML32 buffer. To access these items, FML32 routines must be used; `fml32.h` must be included.

3.3.4 Oracle TSAM Plus Agent Plug-in Implementation

Oracle TSAM Plus Agent plug-in implementation requires the following steps:

- Define “perf_mon_1” in the “e_perf_mon.h” Function Table
- Define the Plug-in Information Variable
- Write the Plug-in Entry Routine
- Writing Concrete Plug-in Implementations

3.3.4.1 Define “perf_mon_1” in the “e_perf_mon.h” Function Table

The following listing shows a `perf_mon_1` defined in the `e_perf_mon.h` function table

Define a "perf_mon_1" defined in "e_perf_mon.h" Function Table

```
static const perf_mon_1 Vtblperfapp_1 = {
       print_app,
       print_svc,
       print_sys,
       print_tran,
};
```

3.3.4.2 Define the Plug-in Information Variable

The following listing shows how to define the plug-in information variable.

Define the Plug-in Information Variable

```
static const _e_pif_plugin_info perf_mon_1_info = {
       { 1, 0 }, /* interface version */
       { 1, 0 }, /* implementation version */
       "abc/tuxedo/tsam", /* implementation id */
       ED_PERF_MON_INTF_ID, /* interface id */
       4, /* virtual table size */
       "ABC, Inc.", /* vendor */
       "Custom Plug-in for Oracle TSAM", /* product name */
       "1.0", /* vendor version */
       EF_PIF_SINGLETON, /* m_flags */
       plugin_destroy,
       plugin_copy
};
```

The changeable members are “implementation version”, “implementation id”, “vendor”, “product name”, “vendor version”. Other items must be kept with same with the sample.

`plugin_destroy` and `plugin_copy` are the general Oracle Tuxedo plug-in routines for destroy and copy. For a Oracle TSAM Plus Plug-in, you can write two empty functions as shown in the listing below:

plugin_destroy and plugin_copy

```
static TM32I _TMDLLENTRY
plugin_destroy (_TCADEF, const struct _e_pif_instance_handles *pIhandles,
TM32U flags)
{
       return(EE_SUCCESS);
}
static TM32I _TMDLLENTRY
plugin_copy (_TCADEF, void *iP,
       const struct _e_pif_interception_data *pInterceptionData,
       struct _e_pif_instance_handles *pIhandles, TM32U flags)
{
       return(EE_SUCCESS);
}
```

3.3.4.3 Write the Plug-in Entry Routine

Each plug-in must have an “entry” routine and specified in plug-in registration process. In this routine, the virtual function table and plug-in information structure must be supplied to the plug-in instance handler.

The following listing shows a plug-in routine example

Plug-in Entry Routine

```
int _TMDLLENTRY
plugin_entry(_TCADEF, const char *pIId,
       const char *pImplId,
       const struct _e_pif_iversion *version,
       const struct _e_pif_data *pData,
       const struct _e_pif_interception_data *pInterceptionData,
       struct _e_pif_instance_handles *pI,
       TM32U flags)
{
const char * const * regData = pData->regdata;
char *logfile = NULL;
       pI->pVtbl = (void *) &Vtblperfapp_1;
       pI->pPluginInfo = (_e_pif_plugin_info *) &perf_mon_1_info;
       pI->pPrivData = NULL;
       return (EE_SUCCESS);
}
```

> **Note**
>
> It is recommended that you use the fixed process shown in the sample. The “entry” routine is called only once to instantiate the plug-in.

3.3.4.4 Writing Concrete Plug-in Implementations

The implementation function table is registered to Oracle Tuxedo in the “entry” routine. Then following chapters will focus on how to write TSAM Plus plug-in based on the corresponding monitoring types.

> **Warning**
>
> Do not make Oracle Tuxedo ATMI calls (except for FML32 operations, `tpalloc`/`tprealloc`/`tpfree` and `tptypes`) in the plug-in. It may result un-expected behavior as Oracle Tuxedo context may be compromised.

- Call Path Monitoring Plug-in Routine
- Service Monitoring Plug-in Routine
- System Server Monitoring Plug-in Routine
- Transaction Monitoring Plug-in Routine

3.3.4.4.1 Call Path Monitoring Plug-in Routine

The call path monitoring plug-in routine is invoked at the monitoring points. For more information, see Oracle TSAM Plus Agent Data Collection Framework

- A Basic Implementation
- Check Commonly Used Metrics
- Generate Call Path Correlation ID

3.3.4.4.1.1 A Basic Implementation

In this example, the routine prints out the passed FML32 buffer:

```
static TM32I _TMDLLENTRY print_app(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL *
monctl, TM32U flags)
{
       Fprint32(*buf);
       return(0);
}
Understanding Current Monitoring Points
```

Call path monitoring is the most comprehensive Oracle Tuxedo application interceptor. It provides a variety of metrics for recording and analysis.

- Determine the monitoring stage
  The monitoring stage itself is a metric with the FML32 field name `TA_MONSTAGE`. The following table lists `TA_MONSTAGE` values.

Table 3-7 TA_MONSTAGE Values

Value	Description
`STMO`	A new call path monitoring is initiated. This is the first record for the current monitored call path.
`ME2Q`	Before a message is sent to the IPC. It could be a request message or reply message. For the monitoring “initiator”, “STMO” replaces “ME2Q” stage since they are at the same point.
`Q2ME`	Before a message is received from the IPC. It could be a request or reply message
`ME2NET`	Before a message sent to the network. It only applies to `GWTDOMAIN`. It could be a request message or reply message.

Table 3-7 (Cont.) TA_MONSTAGE Values

Value	Description
NET2ME	After a message is received from the network. It only applies to GWTDOMAIN. It could be a request message or reply message.

The following listing displays a judge monitoring stage example.

Judge Monitoring Stage

```
{
        char *stage;
        FLDLEN32 len;
        stage = Ffind32(*obuf, TA_MONSTAGE,0,&len);
        if (stage != NULL ) {
                if (strcmp(stage,"STMO") == 0 ) {
                        /* ... */
                }else if (strcmp(stage,"Q2ME" == 0 ) {
                        /* ... */

                /* other processment */
        }
}
```

For “STRING” field type, we recommend to use “Ffind32” routine to get a more fast process.

- Determine the message type
  For an application message transmitted in the Oracle Tuxedo system, it has two choice, request message or reply message. The field TA_MONMSGTYPE indicates the message type.

  The following table lists the TA_MONMSGTYPE values.

Table 3-8 TA_MONMSGTYPE Values

Value	Description
ARQ	Request Message
ARP	Reply Message

- Determine current process location
  The monitoring points always are located in processes of Oracle Tuxedo applications. So understand current process is important. Oracle TSAM Plus framework uses the fields TA_DOMAINID, TA_PID, TA_LMID,TA_MONPROCNAME,TA_GROUPNAME and TA_SRVID (as defined in Table below) to tell the process location.

Table 3-9 Current Process Location Fields

Format	Description
TA_DOMAINID	The domain identifier. Its format is: domainid: masternname:ipckey • domainid is the DOMAINID configured in UBBCONFIG. If it is not set, TUXDOM is used. • Mastername is the master machine name. • Ipckey is the key in UBBCONFIG

Table 3-9 (Cont.) Current Process Location Fields

Format	Description
TA_PID	Process ID
TA_LMID	Logic machine ID
TA_MONPROCNAME	Process name.
TA_GROUPNAME	Oracle Tuxedo server group name
TA_SRVID	Oracle Tuxedo server ID.

> **Note**
>
> Not all metrics available for certain processes. For example, for client processes, TA_SRVID is not available.

3.3.4.4.1.2 Check Commonly Used Metrics

After get the necessary information on the monitoring stage, message type and process location, the next step is to check the common used metrics also carried in the FML32 buffer. The metrics will be available depending on the conditions mentioned previously.

The following table lists the commonly used metrics

Table 3-10 Commonly Used Metrics

Field Name	Type	Description	Stage
TA_MONCORRID	string	The correlation ID of this monitored call path	All
TA_MONMSGSIZE	long	The message size of current message	All
TA_MONMSGQUEUED	long	How many message queued on the server request IPC queue	Request Message Q2ME
TA_MONSTARTTIMESEC	long	The second part of timestamp when this call path monitoring is initiated. It is the number of seconds since epoch.	All
TA_MONSTARTTIMEUSEC	long	The microsecond part of the startup timestamp. It is always with TA_MONSTARTUTIMESEC to provide a more fine-grained time measurement.	All

Table 3-10 (Cont.) Commonly Used Metrics

Field Name	Type	Description	Stage
TA_MONLASTTIMESEC	long	The second part of timestamp when the monitored message entering/leaving a transport. It is the number of seconds since epoch. A transport is the way carrying message, such as IPC queue and network. A typical usage is, • When a request is fetched from IPC queue, the TA_LASTTIMESEC indicates the timestamp when the request message was put into queue. • When a request is fetched from network, the TA_LASTTIMESEC indicates the timestamp when the peer process sent the message to network.	All
TA_MONLASTTIMEUSEC	long	The microsecond part of the last time timestamp. It is always with TA_MONLASTTIMESEC to provide a more fin-grained time measurement.	All
TA_MONLGTRID	string	The GTRID of current monitoring points if the call path involved in transaction.	Monitoring points involved transaction
TA_MONCLTADDR	string	The remote client address. If the monitoring is started from an Oracle Tuxedo workstation client, WSH, JSH or GWWS, TSAM Plus framework will attach the client ip address and port number to call path information propagation. The format is //ip address:port.	All
TA_MONDEPTH	short	The call path depth. A hop from one service to another is deemed the depth increased one. The start value at the initiator is 0.The detail can be referred at TSAM Plus User Guide.	All
TA_MONERRNO	long	The error number set by Oracle Tuxedo infrastructure.	Reply Message
TA_MONURCODE	long	The urcode of tpreturn.	Reply Message
TA_MONSVCNAME	string	The service name of current monitoring points involved. For request message, it is the target service name and for reply message, it is the service which returns the reply.	All
TA_MONHOSTSVC	string	The service name of current service routine	Monitoring points in a application server.
TA_MONCALLFLAG	long	The call flags set in tpcall/tpacall	Request Message ME2Q STMO

Table 3-10 (Cont.) Commonly Used Metrics

Field Name	Type	Description	Stage
TA_MONCALLMODE	short	The call type, 1 - tpacall, 2 - tpcall, 3 - tpforward	Request Message ME2Q STMO
TA_MONQID	string	The request queue id of server which provides current service. Its format is “physical queue key - Oracle Tuxedo logic queue name”. For example, 14444547-00004.00018	Request Message Q2ME
TA_MONLDOM	string	The local domain configuration. Its format is ldom:domainid. For example DOM1:FINANCE. The detail information of the “LDOM” and “DOMAINID” can be referred Oracle Tuxedo Manual of DMCONFIG.	ME2NET NET2ME
TA_MONRDOM	string	The remote domain configuration. Its format is same with TA_MONLDOM but the values are for remote domain.	ME2NET NET2ME
TA_MONWSENDPOINT	string	The web service end point URL of GWWS.	Reply Message ME2Q
TA_MONCPUTIME	long	CPU time used for service execution.	Reply Message ME2Q

> **Note**
>
> For some self-describe buffer types, such as STRING, the size might be zero.

3.3.4.4.1.3 Generate Call Path Correlation ID

The correlation ID must be given by the plug-in at the monitoring initiating stage, which is the TA_MONSTAGE value is “STMO”. The Oracle TSAM Plus framework sets PI_CORRID_REQUIRED in the MONITORCTL mon_flag.

If no correlation ID is given, an error is reported. The Oracle TSAM Plus default plug-in provides the correlation ID also. The following two scenarios need to be considered:

- Working with the Oracle TSAM Plus default plug-in.
  The custom plug-in can skip the correlation ID generation. If the custom plug-in wants to overwrite the correlation ID generated by the Oracle TSAM Plus default plug-in, the interceptor sequence of custom plug-in must come after the Oracle TSAM Plus default plug-in.
- Working without The Oracle TSAM Plus default plug-in
  If the Oracle TSAM Plus default plug-in is removed from the Oracle Tuxedo plug-in framework, the custom plug-in must supply the correlation ID i. For example:

```
if (monctl->mon_flag & PI_CORRID_REQUIRED) {
       strcpy(monctl->corr_id, mygetid());
}
```

"mygetid()" is an assumed ID generation routine. The length of the new ID must not exceed the size of corr_id of MONITORCTL.

To help ID generation, the custom plug-in can use a Oracle TSAM Plus framework service to get a correlation ID. The listing below displays an ID generation example.

ID Generation Example

```
extern int _TMDLLENTRY tmmon_getcorrid(char *obuf, int len);
...
if (monctl->mon_flag & PI_CORRID_REQUIRED) {
       char new_corrid[MAXCORRIDLEN];
       if (tmmon_getcorrid(new_corrid, sizeof(new_corrid)) == 0 ) {
              strcpy(monctl->corr_id,new_corrid);
       }
}
...
```

> **Note**
>
> When using the Oracle TSAM Plus framework correlation ID generation routine, libtsam must be linked with the plug-in.

3.3.4.4.2 Service Monitoring Plug-in Routine

Service monitoring is a straightforward procedure. The data collection points are before and after the service routine invocation. The plug-in is invoked when a request is to be executed and a reply to be sent back to client.

- A Basic Implementation
- Check Commonly Used Metrics

3.3.4.4.2.1 A Basic Implementation

In this example, the routine prints out the passed FML32 buffer:

```
static TM32I _TMDLLENTRY print_svc(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL *
monctl, TM32U flags)
{
       Fprint32(*buf);
       return(0);
}
```

3.3.4.4.2.2 Check Commonly Used Metrics

The following table lists the service monitoring plug-in routine metrics.

Table 3-11 Service Monitoring Plug-in Routine Metrics

Field Name	Type	Description
TA_MONMSGWAITTIME	long	The request message waiting time in server's request IPC queue before execution. The unit is millisecond. The waiting time is computed in two scenarios, • Oracle Tuxedo 12cR1 and later is the request sender The waiting time is computed by considering the last time stamp of transport to this service. The waiting time is exact. • Pre-Oracle Tuxedo 12cR1 release sender. The waiting time is computed based on average queue length and last service execution time and the dispatching thread number. This is an approximate value. It only applies to a server which provides similar services and the execution time is steady.
TA_MONMSGSIZE	long	The message size of reply message.
TA_MONMSGQUEUED	long	The number of messages queued on the server request IPC queue currently.
TA_MONLASTTIMESEC	long	The number of seconds since epoch when the service begin to execute
TA_MONLASTTIMEUSEC	long	The microsecond part of the timestamp. It is used with TA_MONLASTTIMESEC
TA_MONERRNO	long	Oracle Tuxedo return error code, that is tperrno
TA_MONURCODE	long	The urcode of tpreturn.
TA_MONEXECTIME	long	The response time in millisecond of current service execution. It is computed by the Oracle TSAM Plus framework. Plug-in can also get the current time and the last time timestamp.
TA_MONCPUTIME	long	How much CPU time is used of current service execution. It is in milliseconds.
TA_MONSVCNAME	string	The service name.
TA_MONLOCATION	string	The process location of current process. It has same meaning in call path monitoring.

3.3.4.4.3 System Server Monitoring Plug-in Routine

Oracle TSAM Plus supports several types of Oracle Tuxedo system servers monitoring: GWTDOMAIN, BRIDGE, and GWWS. The monitoring focus on the throughput, outstanding request number and message number queued on network. The plug-in is invoked periodically by the Oracle TSAM Plus framework. The interval is specified by the monitoring policy. Data collection occurs on the on-going server operations.

- A Basic Implementation
- Check Commonly Used Metrics

3.3.4.4.3.1 A Basic Implementation

In this example, the routine prints out the passed FML32 buffer:

```
static TM32I _TMDLLENTRY print_sys(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL *
monctl, TM32U flags)
{
       Fprint32(*buf);
       return(0);
}
```

3.3.4.4.3.2 Check Commonly Used Metrics

The following table lists the system server monitoring plug-in routine metrics.

Table 3-12 System Server Monitoring Plug-in Routine Metrics

Field Name	Type	Description
TA_MONLINKNUM	short	The number of network link connected to current server. If the value is more than 1, then the following statistics data on network link are in FML occurrences style. For example, TA_MONLINKADDR[0] is belong to the first network link, TA_MONLINKADDR[1] is belong to the second network link etc.
TA_MONLINKSTATUS	short	The status of the network link, three possible values, 1 - initialize stage. 0 - connected and is ok. -1 connection lost.
TA_MONNUMPEND	long	The number of messages queued on network buffer for this network link. The buffer is for Oracle Tuxedo network layer instead of system network stack. This is a snapshot value reflecting the number situation when plug-in is invoked.
TA_MONBYTESPEND	long	The number of messages bytes queued on network buffer. It is related with TA_MONNUMPEND but computing the data volume
TA_MONNUMWAITRPLY	long	The outstanding request number on this network link. That means how many request message are waiting for reply. It only applies to GWTDOMAIN. BRIDGE does not support this metric. This is a snapshot value.
TA_MONACCNUM	long	The accumulated message number for this network link between current plug-in invocation and last plug-in invocation which controlled by the “sysinterval” policy. This is a throughput value reflecting the accumulated information between an interval.
TA_MONACCBYTES	long	The accumulated message bytes. It is related TA_MONACCNUM but computing the data volume. This is a throughput value.
TA_MONLINKADDR	string	The link address, for GWTDOMAIN, it is the RDOM defined in UBBCONFIG. For BRIDGE, it is the remote host name.
TA_MONINRPCFAIL	long	The number of RPC failed requests for inbound of GWWS.

Table 3-12 (Cont.) System Server Monitoring Plug-in Routine Metrics

Field Name	Type	Description
TA_MONOUTRPCFAIL	long	The number of RPC failed request for outbound of GWWS.
TA_MONINBOUNDPEND	long	The number of pending request waiting for reply for inbound of GWWS.
TA_MONOUTBOUNDPEND	long	The number of pending request waiting for reply for outbound of GWWS

3.3.4.4.4 Transaction Monitoring Plug-in Routine

Oracle TSAM Plus also traces critical routines invocation in XA transaction. The scope includes tpbegin,tpcommit, tpabort,xa_xxx calls and GWTDOMAINS transaction routines.

- A Basic Implementation
- Check Commonly Used Metrics
- Configure the Plug-in to Oracle Tuxedo

3.3.4.4.4.1 A Basic Implementation

In this example, the routine prints out the passed FML32 buffer:

```
static TM32I _TMDLLENTRY print_tran(perf_mon_1 * ip,FBFR32 **buf, MONITORCTL
* monctl, TM32U flags)
{
       Fprint32(*buf);
       return(0);
}
```

3.3.4.4.4.2 Check Commonly Used Metrics

The following lists the commonly used transaction monitoring plug-in routine metrics.

Table 3-13 Transaction Monitoring Plug-in Routine Metrics

Field Name	Type	Description
TA_MONXANAME	string	The routine name of a XA transaction, such as “tpbegin”, “xa_commit” etc.
TA_MONXACODE	long	The routine return code
TA_MONEXECTIME	long	The routine execution time in millisecond.
TA_MONRMID	long	The resource manager instance ID. It only applies to xa_xxx calls
TA_MONLGTRID	string	The global transaction ID of current transaction
TA_MONRGTRID	string	The parent transaction’s GTRID. It only applies to GWTDOMAIN when it is a network subordinator.
TA_MONLOCATION	string	The process location of current process. It has same meaning in call path monitoring.

3.3.4.4.4.3 Configure the Plug-in to Oracle Tuxedo

> **Note**
>
> The plug-in will run in Oracle Tuxedo infrastructure. It must be well tested before configure to Oracle Tuxedo production environment.

- Register to Oracle Tuxedo
- Un-register from Oracle Tuxedo

3.3.4.4.4.3.1 Register to Oracle Tuxedo

Oracle Tuxedo uses the `epifreg` command to register the plug-ins to the Oracle Tuxedo registry so that the infrastructure can invoke the plug-in at run time. Oracle TSAM Plus uses the Oracle TSAM Plus framework to invoke the plug-in.

The following listing shows how the `epifreg` command is used to invoke a plug-in.

Using epifreg to Invoke a Plug-in

```
epifreg -r -p abc/tuxedo/tsam -i engine/performance/monitoring \
       -o SYSTEM -v 1.0 -f /test/abc/customplugin.so -e plugin_entry
epifregedt -s -k "SYSTEM/impl/bea/performance/monfan" \
       -a InterceptionSeq=bea/performance/monshm \
          -a InterceptionSeq=abc/tuxedo/tsam
```

In this, there are two steps required to register the custom plug-in Oracle Tuxedo.

1. Using “epifreg” to register the custom implementation to Oracle Tuxedo.
   a. “`-p`” option specifies the implementation id and it must be consistent the value specified in source code.
   b. “`-v`” indicates the version number.
   c. “`-f`” specifies the dynamic library path.
   d. “`-e`” specifies the “entry” routine described in the “General Steps” section.
2. Using “`epifregedt`” to change the fan-out plug-in “`InterceptionSeq`” attribute.
   Oracle TSAM Plus supports a Fan-out plug-in mechanism which means multiple plug-ins can work together. Oracle TSAM Plus Agent provides the Fan-out plug-in and a default interceptor plug-in. The custom plug-in is an additional interceptor plug-in.

   The “-a `InterceptionSeq=xxx`” option tells the Fan-out plug-in invokes the interceptor plug-in using the specified order. “xxx” is the implementation id. In this example, the Oracle Tuxedo default interceptor plug-in implementation ID, “`bea/performance/monshm`”, is invoked before the custom plug-in implementation ID “`abc/tuxedo/tsam`”.
3. If you have multiple custom plug-in developed, you need to register them first with “epifreg”, then modify the invocation sequence with “`epifregedt`” with the proper “`InterceptionSeq`” sequence.

3.3.4.4.4.3.2 Un-register from Oracle Tuxedo

“`epifunreg`” can be used to un-register a specified plug-in, for example,

```
epifunreg -p abc/tuxedo/tsam
```

After unregistering the custom plug-in, you must use “`epifregedt`” to modify the Fan-out plug-in invocation again based on current available plug-ins. For example:

```
epifregedt -s -k "SYSTEM/impl/bea/performance/monfan" \
-a InterceptionSeq=bea/performance/monshm
```

> **Note**
>
> It is strongly recommended to register/unregister/modify the plug-in after shutting down an Oracle Tuxedo application.

3.3.5 Oracle TSAM Plus Agent Plug-in Development/Deployment Notes

- Do not use Oracle Tuxedo ATMI calls in the plug-in except for the FML32 operations `tpalloc`/ and `tptypes`. The monitoring points are embedded in the Oracle Tuxedo communication framework. Embedded ATMI calls may compromise current Oracle Tuxedo context.
- You cannot free FML32 buffers passed by the plug-in.
- If there is any information returned to the Oracle TSAM Plus framework, such as new correlation ID, the latest plug-in changes take effect.
- Do not change the MONITORCTL udata. It is a read only interception of application messages. Any modification will result un-expected behavior.