

Oracle® Tuxedo Systems and Application Monitor (TSAM)

Plug-in Programming Guide

10g Release 3 (10.3)

January 2009

Copyright © 2007, 2009, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

1. Oracle TSAM Agent Plug-in Programming Introduction	
Overview	1-1
2. Oracle TSAM Agent Data Collection Framework	
Overview	2-1
3. Creating an Oracle TSAM Agent Custom Plug-in	
Overview	3-1
Tuxedo Plug-in Framework Concepts	3-1
Interface	3-2
Implementation	3-2
Plug-in Register/Un-register/Modifications	3-4
Developing a Oracle TSAM Agent Plug-in	3-4
Create Plug-in Source Code	3-4
Build the Plug-in	3-8
Register the Plug-in	3-8
Enable Oracle TSAM Monitoring	3-9
Run a Call and Check the Standard Output.	3-9
Oracle TSAM Agent Plug-in Interface	3-10
Version and Interface Identifier	3-10
Function Table	3-13
Field Map Operation Facility	3-14
Other Help Header Files	3-15

Oracle TSAM Agent Plug-in Implementation	3-15
Define “perf_mon_1” in the “e_perf_mon.h” Function Table.	3-15
Define the Plug-in Information Variable	3-16
Write the Plug-in Entry Routine	3-17
Writing Concrete Plug-in Implementations	3-18
Call Path Monitoring Plug-in Routine	3-18
A Basic Implementation	3-18
Understanding Current Monitoring Points	3-18
Check Commonly Used Metrics.	3-21
Change Required Fields	3-23
Generate Call Path Correlation ID	3-24
Service Monitoring Plug-in Routine	3-25
A Basic Implementation	3-25
Check Commonly Used Metrics.	3-26
System Server Monitoring Plug-in Routine.	3-27
A Basic Implementation	3-27
Check Commonly Used Metrics.	3-28
Transaction Monitoring Plug-in Routine	3-29
A Basic Implementation	3-29
Check Commonly Used Metrics.	3-29
Configure the Plug-in to Tuxedo	3-30
Register to Tuxedo	3-30
Un-register from Tuxedo	3-31
Oracle TSAM Agent Plug-in Development/Deployment Notes	3-31

Oracle TSAM Agent Plug-in Programming Introduction

This topic contains the following sections:

- [Overview](#)

Overview

The Oracle TSAM Agent includes three major layered modules:

- Oracle TSAM framework

The Oracle TSAM 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 Agent ships with a default plug-in. The default plug-in performs event trigger evaluation and sends metrics to LMS

- LMS (local monitor server)

The LMS synchronizes data with the Oracle TSAM Manager.

The Oracle TSAM Agent and Oracle TSAM Manager provide a complete solution for data collection, aggregation, storage and presentation. To support various requirements for monitoring data usage, the Oracle TSAM 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 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

Oracle TSAM Agent Data Collection Framework

This topic contains the following sections:

- [Overview](#)

Overview

The Oracle TSAM Agent enhances Tuxedo infrastructure to collect the performance metrics when TSAM is enabled. The instrument covers the major performance sensitive areas in Tuxedo applications, that is call path stages, services, transactions and system servers. TSAM Agent uses Tuxedo FML32 typed buffer 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 Tuxedo ATMI applications. [Listing 2-1](#) lists the monitoring points.

Table 2-1 Call Path Monitoring Points

Stage	Supported Tuxedo Process Types
Before request message sent to IPC queue	Native Client, Application Server, GWTDOMAN, BRIDGE
Request sent to IPC queue failure	Native Client, Application Server, WSH and JSH

Table 2-1 Call Path Monitoring Points

After request message got from IPC queue	Application Server, GWTDOMAIN
Before reply message sent to IPC queue	Application Server, GWTDOMAIN, BRIDGE
After reply message got from IPC queue	Native Client, Application Server, GWTDOMAIN
Before request message sent to network	GWTDOMAIN
After request message got from network	GWTDOMAIN
Before reply message sent to network	GWTDOMAIN, WSH, JSH
After reply message got from network	GWTDOMAIN

Table 2-2 Service Monitoring Points

Stage	Supported Tuxedo Process Types
After request ¹ message got from IPC queue	Application Server, GWTDOMAIN
Before reply message sent to IPC queue	Application Server, GWTDOMAIN

1. Only data collection point, no plug-in invocation

Table 2-3 System Server Monitoring Points

Stage	Supported Tuxedo Process Types
Main Loop ¹	GWTDOMAIN, BRIDGE

1. The metrics are collected internally and this point is to pass the data to plug-in

Table 2-4 Transaction Monitoring Points

Stage	Supported Tuxedo Process Types
Before the ¹ transaction routine executed	Native Client, Application Server, TMS, GWTDOMAIN, WSH, JSH, TMQFORWARD
After the transaction routine executed	Native Client, Application Server, TMS, GWTDOMAIN, WSH, JSH, TMQFORWARD

1. Only data collection point, no plug-in invocation

Note: The monitoring point is not necessarily added for all the message processing stages. It depends on the process internal running model. For example, when BRIDGE or WSH receives a message from the IPC queue, the message is forwarded to the network immediately. In this case, there is only one monitoring point (point 1).

Creating an Oracle TSAM Agent Custom Plug-in

This topic contains the following sections:

- [Overview](#)
- [Developing a Oracle TSAM Agent Plug-in](#)
- [Oracle TSAM Agent Plug-in Interface](#)
- [Oracle TSAM Agent Plug-in Implementation](#)
- [Oracle TSAM Agent Plug-in Development/Deployment Notes](#)

Overview

Tuxedo has a built-in plug-in framework that facilitates additional functionality. For example, the Tuxedo security mechanism is constructed on the plug-in framework. 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 a Tuxedo ATMI service. Oracle TSAM Agent also use the Tuxedo plug-in framework to attach different data receivers.

Tuxedo Plug-in Framework Concepts

The following section highlights Tuxedo plug-in framework key concepts.

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 Tuxedo plug-in framework and uses the following format:

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

The Oracle TSAM 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**

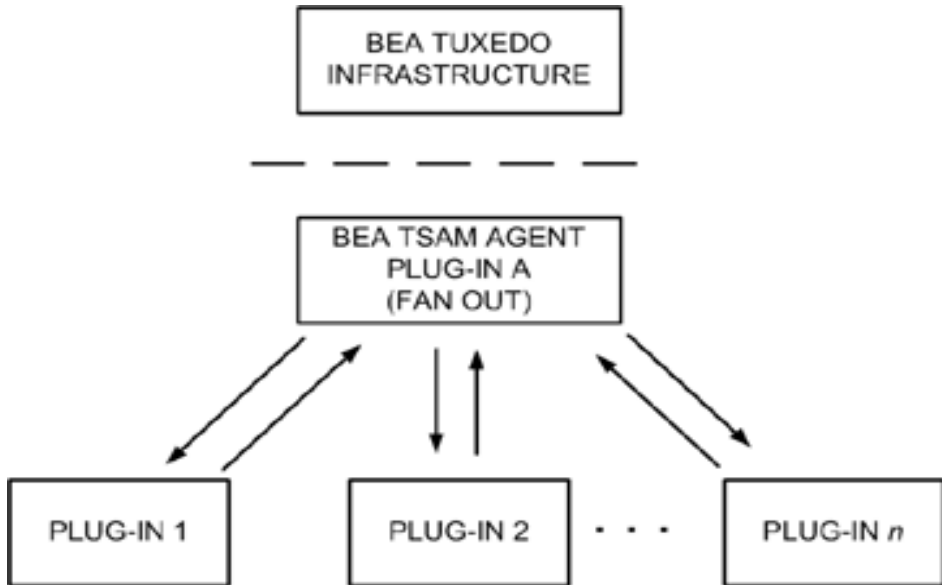
The data structure defines the concrete information conveyed between plug-in caller and implementation. The function declaration defines the routines must be implemented by plug-in.

Implementation

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

Tuxedo supports two types of interceptors: Fan-out interceptors and Stack interceptors. The Oracle TSAM Agent uses the Fan-out interceptors. [Figure 3-1](#) displays the Oracle TSAM Agent plug-in architecture.

Figure 3-1 Oracle TSAM Agent Plug-in Architecture



When the 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 method X 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 Agent provides the Fan-out plug-in which allows you to write/create an interceptor plug-in.

Plug-in Register/Un-register/Modifications

Once the plug-in is written it must be registered in the Tuxedo registry so that the functional components will locate the plug-in and invoke the appropriate methods. 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

Developing a Oracle TSAM Agent Plug-in

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

A simple Oracle TSAM 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 Agent default plug-in.

1. [Create Plug-in Source Code](#)
2. [Build the Plug-in](#)
3. [Register the Plug-in](#)
4. [Enable Oracle TSAM Monitoring](#)
5. [Run a Call and Check the Standard Output.](#)

Create Plug-in Source Code

[Listing 3-1](#) displays an example of the Oracle TSAM plug-in `customplugin.c`.

Listing 3-1 Oracle TSAM Agent `customplugin.c` Plug-in Source Code Example

```
#include <e_pif.h>
#include <tpadm.h>
#include <fm132.h>
```

```

#include <e_perf_mon.h>

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

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

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

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

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

static TM32I _TMDLLENTY 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 _TMDLLENTY
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 _TMDLLENTY
plugin_destroy (_TCADEF, const struct _e_pif_instance_handles *pIhandles,
               TM32U flags)
{
    return(EE_SUCCESS);
}

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

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

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

static TM32I _TMDLLENTY print_tran(perf_mon_1 * ip,FBFR32 **buf,

```

```
MONITORCTL * monctl, TM32U flags)
{
    Fprint32(*buf);
    return(0);
}
```

Build the Plug-in

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

Register the Plug-in

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

1. Shutdown your tuxedo application by “tmshutdown”
2. Compose a shell script named “reg.sh”
3. Run the script

```
sh ./reg.sh
```

4. Boot your Tuxedo applications by "tmboot"

[Listing 3-2](#) displays an example of the reg.sh shell script

Listing 3-2 reg.h 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/mongui \
-a InterceptionSeq=abc/tuxedo/tsam \
```

Enable Oracle TSAM Monitoring

Enable TSAM Monitoring using the TMMONITOR environment variable: `tmadmin->chmo` or by using the Oracle TSAM console

For more information, see the [Oracle TSAM Administration Guide](#).

Run a Call and Check the Standard Output.

You will find the metrics collected printed out.

[Listing 3-3](#) displays the metrics print out.

Listing 3-3 Metrics Print Out Example

```
TA_MONDEPTH          1
TA_MONPROCTYPE       2
TA_MONMSGSIZE        6
TA_MONMSGQUEUED      1
TA_MONLASTTIMESEC    1189759850
TA_MONLASTTIMEUSEC   730754
TA_MONSTARTTIMESEC   1189759850
TA_MONSTARTTIMEUSEC  730754

TA_MONCORRID         JOLTDOM:bjsol18:72854 SITE1 client 9597 1 1
TA_MONMSGTYPE        ARQ
TA_MONSTAGE          Q2ME
TA_MONLOCATION        JOLTDOM:bjsol18:72854 SITE1 GROUP2 simpserv 18 9588
TA_MONSVCNAME        TOUPPER
TA_MONHOSTSVC        TOUPPER
TA_MONSVCSEQ         client-TOUPPER-18218-0
TA_MONPSVCSEQ        INITIATOR
TA_MONQID            14441475-00004.00018

TA_MONDEPTH          1
TA_MONPROCTYPE       2
TA_MONMSGSIZE        6
TA_MONLASTTIMESEC    1189759850
TA_MONLASTTIMEUSEC   730754
```

```
TA_MONSTARTTIMESEC      1189759850
TA_MONSTARTTIMEUSEC     730754
TA_MONERRNO             0
TA_MONURCODE            0
TA_MONCORRID            JOLTDOM:bjsol18:72854 SITE1 client 9597 1 1|
TA_MONMSGTYPE           ARP
TA_MONSTAGE             ME2Q
TA_MONLOCATION           JOLTDOM:bjsol18:72854 SITE1 GROUP2 simpserv 18 9588
TA_MONSVCNAME           TOUPPER
TA_MONHOSTSVC           TOUPPER
TA_MONSVCSEQ            client-TOUPPER-18218-0
TA_MONPSVCSEQ           INITIATOR
```

Oracle TSAM Agent Plug-in Interface

All Oracle TSAM Plug-in interface contents are defined in the `$TUXDIR/include/e_perf_mon.h` file. When you build a Oracle TSAM 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](#)
- [Field Map Operation Facility](#)

Version and Interface Identifier

[Listing 3-4](#) provides a version and identifier example.

Listing 3-4 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
```

[Listing 3-5](#) displays the Oracle TSAM framework and plug-in core data structure.

Listing 3-5 Core Data Structure

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

Table 3-1 MONITORCTL Members

Members	Description
fieldsmap	Indicates which metrics field is available in the FML32 buffer passed from TSAM framework. Once it receives information from the TSAM Framework, is returns any changed or updated information if the plug-in makes changes to the required fields.
monitoring_policy	Internal use only
corr_id	It is used to bring the corraling ID from plug-in to TSAM 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 framework and plug-in to indicate the requirement and changes.

[Table 3-2](#) lists the MONITORCTL array size definitions.

Table 3-4 lists the mon_flag Values.

Table 3-2 MONITORCTL Array Size Definitions

Array	Size	Description
<code>/* Size of fieldsmap*/</code>	<code>#define MAXMAPSIZE</code>	128
<code>/* Size of monitoring_pol icy */</code>	<code>#define MAXPOLICYLEN</code>	128
<code>/* Size of corr_id*/</code>	<code>#define MAXCORRIDLEN</code>	256

Table 3-3 mon_flag Values

Members	Description
<code>#define PI_CORRID_REQU IRED 0x00000001</code>	PI_CORRID_REQUIRED is set by TSAM 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.
<code>#define PI_EDITABLE_FI ELDS 0x00000002 */</code>	PI_EDITABLE_FIELDS is set by TSAM framework when the required fields of TMMONITOR specification is editable by the plug-in.
<code>#define PI_EDITABLE_PO LICY 0x00000004</code>	PI_EDITABLE_POLICY internal use only.
<code>#define PI_UPDATED_FIE LDS 0x00000008</code>	PI_UPDATED_FIELDS is set by the plug-in if PI_EDITABLE_FIELDS is set by TSAM framework and the plug-in is also changed the required fields by fieldsmap of MONITORCTL. TSAM framework will check this to update the data collection engine.

Table 3-3 mon_flag Values

Members	Description
#define PI_UPDATED_POL ICY 0x00000010	PI_UPDATED_POLICY is internal use only.
#define PI_NOCALL 0x00000020	PI_NOCALL is internal use only.

Function Table

[Listing 3-6](#) defines the plug-in implementation methods.

Listing 3-6 Function Table

```
typedef struct perf_mon_1_Vtbl {
    TM32I (_TMDLLENTY *_ec_perf_mon_app) _((
        struct perf_mon_1_Vtbl * ip,
        FBFR32 **buf,
        MONITORCTL *mon_ctl,
        TM32U flags
    ));
    TM32I (_TMDLLENTY *_ec_perf_mon_svc) _((
        struct perf_mon_1_Vtbl * ip,
        FBFR32 **buf,
        MONITORCTL *mon_ctl,
        TM32U flags
    ));
    TM32I (_TMDLLENTY *_ec_perf_mon_sys) _((
        struct perf_mon_1_Vtbl * ip,
        FBFR32 **buf,
        MONITORCTL *mon_ctl,
        TM32U flags
    ));
    TM32I (_TMDLLENTY *_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 of this function table. **Custom plug-in need not handle this.**
- `FBFR32 **buf`
the address of the metrics buffer in FML32 type.
- `MONITORCTL *mon_ctl`
the control structure.
- `TM32U flags`
the bits flag in a 32-byte, unsigned integer.

Field Map Operation Facility

The **fieldsmap** contains the quick reference of which metrics are stored in the **buf** parameter of the interface methods. A set of operation facilities are designed to check/set the **fieldsmap** data.

```
#define BASENUM          30002401
```

BASENUM is the base number of TSAM FML32 metrics fields. The performance metrics are defined as Tuxedo built-in FML32 fields which are included `$TUXDIR/include/tpadm.h`.

```

#define IS_FIELDSET(FIELDSMAP, FIELDID)      ( FIELDSMAP[Fldno32(FIELDID)
- BASENUM ] )

#define SET_FIELD(FIELDSMAP, FIELDID)      { FIELDSMAP[Fldno32(FIELDID)
- BASENUM ] = 1; }

#define UNSET_FIELD(FIELDSMAP, FIELDID)    { FIELDSMAP[Fldno32(FIELDID)
- BASENUM ] = 0; }

```


IS_FIELDSET is used to determine a field is set or not in a “**fieldsmap**”. **SET_FIELD** is used to set a field to a “**fieldsmap**” and **UNSET_FIELD** is to clear a field set in a “**fieldsmap**”. These operation facilities can be used in the plug-in if it needs to relay the required field changes to the Oracle TSAM framework.

Other Help Header Files

- `$TUXDIR/include/e_pif.h`

It is the Tuxedo general plug-in definition file. It must be included in the plug-in source code.

- `$TUXDIR/include/tpadm.h`

It is the Tuxedo built-in FML32 fields definition files. All performance metrics are defined as FML32 fields.

- `$TUXDIR/include/fml32.h`

The metrics collected are stored in a Tuxedo FML32 buffer. To access these items, FML32 routines must be used. So the “`fml32.h`” must be included.

Oracle TSAM Agent Plug-in Implementation

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

1. [Define “perf_mon_1” in the “e_perf_mon.h” Function Table](#)
2. [Define the Plug-in Information Variable](#)
3. [Write the Plug-in Entry Routine](#)

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

[Listing 3-7](#) shows a `perf_mon_1` defined in the `e_perf_mon.h` function table example.

Listing 3-7 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,  
};
```

Define the Plug-in Information Variable

[Listing 3-8](#) shows how to define the plug-in information variable.

Listing 3-8 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 Tuxedo plug-in routines for destroy and copy. For a Oracle TSAM Plug-in, you can write two empty functions as shown in [Listing 3-9](#).

Listing 3-9 `plugin_destroy` and `plugin_copy`

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

```

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

```

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.

[Listing 3-10](#) displays a plug-in routine example.

Listing 3-10 Plug-in Entry Routine

```

int _TMDLLENTY
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 not to use the fixed process shown in the sample. The “entry” routine is called only once to instantiate the plug-in.

Writing Concrete Plug-in Implementations

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

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

Call Path Monitoring Plug-in Routine

The call path monitoring plug-in routine are invoked at the monitoring points. For more information, see [“Oracle TSAM Agent Data Collection Framework” on page 2-1](#).

A Basic Implementation

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

```
static TM32I _TMDLLENTY 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 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`. [Table 3-4](#) lists `TA_MONSTAGE` values.

Table 3-4 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.
NET2ME	After a message is received from the network. It only applies to GWTDOMAIN. It could be a request message or reply message.

[Listing 3-11](#) displays a judge monitoring stage example.

Listing 3-11 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 Tuxedo system, it has two choice, request message or reply message. The field TA_MONMSGTYPE indicates the message type.

Table 3-5 TA_MONMSGTYPE Values

Value	Description
ARQ	Request Message
ARP	Reply Message

- Determine current process location

The monitoring points always are located in processes of Tuxedo applications. So understand current process is important. TSAM framework uses the field TA_MONLOCATION to tell the plug-in the process location of current monitoring point. The format of TA_MONLOCATION is different for Tuxedo client process and server process. The major goal is to provide enough information to locate the process uniquely in this Tuxedo domain.

Table 3-6 TA_MONLOCATION Format

Format	Description
Client Format	DOMAINID:master hostname:IPCKEY LMID processname processid Example: JOLTDOM:bjsol18:72854 SITE1 client 15391
Server Format	DOMAINID:master hostname:IPCKEY LMID group processname serverid processid Example: JOLTDOM:bjsol18:72854 SITE1 GROUP2 simpserv 18 9704
WSH/JSH ¹	DOMAINID:master hostname:IPCKEY LMID group processname processid Example: JOLTDOM:bjsol18:72854 SITE1 JOLTGRP JSH 9904

1. The group of WSH/JSH is the group of its listeners, that is WSL/JSL

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 above.

Table 3-7 TA_MONLOCATION Format 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 ¹
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
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, <ul style="list-style-type: none"> • 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

Table 3-7 TA_MONLOCATION Format Metrics

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 Tuxedo workstation client, WSH, JSH or GWWS, TSAM 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 User Guide.	All
TA_MONERRNO	long	The error number set by 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 3-7 TA_MONLOCATION Format Metrics

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 - Tuxedo logic queue name”. For example, 14444547-00004.00018	Request Message Q2ME
TA_MONLDDOM	string	The local domain configuration. Its format is ldom:domainid. For example DOM1:FINANCE. The detail information of the “LDDOM” and “DOMAINID” can be referred Tuxedo Manual of DMCONFIG.	ME2NET NET2ME
TA_MONRDOM	string	The remote domain configuration. Its format is same with TA_MONLDDOM but the values are for remote domain.	ME2NET NET2ME
TA_MONWSENDPOINT	string	The web service end point URL of GWWS.	Reply Message ME2Q

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

Change Required Fields

Oracle TSAM Agent allows to reduce the metrics collected by specifying the required fields of TMMONITOR specification. TMMONITOR syntax consists of three parts: monitoring type, monitoring policy and required fields. They are separated by a colon. The usage of monitoring type and policy can be referred at TSAM User Guide. The required fields supported by call path monitoring and service monitoring.

- Call Path Monitoring

```
appfields=field1,field2,field3...
```

- Service Monitoring

```
svcfields=field1,field2,field3...
```

Note: The required fields option is only for custom plug-ins. It cannot apply to the Oracle TSAM default plug-in. If required fields are used, the Oracle TSAM default plug-in will not function normally, which means the Oracle TSAM Manager cannot function normally. It also means you cannot use the Oracle TSAM default plug-in together with custom plug-ins.

Normally, using required fields is not recommended.

The default behavior is all the available metrics are collected by the Oracle TSAM framework.

If required fields are specified, only the specified and basic information is available in the metrics buffer.

To enable the TMMONITOR required fields, set TMMONITOR specifications in the environment variable or `tmadmin`. For example:

```
export
TMMONITOR=app::appfields=TA_MONSVNAME,TA_MONSTAGE,TA_MONLOCATION
```

To change the required fields, the plug-in must see whether the `mon_flag` of `MONITORCTL` allows to change it.

```
if (monctl->monflag & PI_EDITABLE_FIELDS ) {
    UNSET_FIELD(monctl->fieldsmap,TA_MONLOCATION);
    SET_FIELD(monctl->fieldsmap,TA_MONMSGSIZE);
    SET_FIELD(monctl->fieldsmap,TA_MONLDOM);
    monctl->monflag |= PI_UPDATED_FIELDS;
}
```

The `MONITORCTL monflag` must be set to `PI_UPDATED_FIELDS` to tell the Oracle TSAM framework to update the required fields. This change impacts all the monitoring points on the call path.

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 framework sets `PI_CORRID_REQUIRED` in the `MONITORCTL mon_flag`. If no correlation ID is given, an error is reported. The Oracle TSAM default plug-in provides the correlation ID also. Two scenarios need to consider,

- Working with the Oracle TSAM 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 default plug-in, the interceptor sequence of custom plug-in must come after the Oracle TSAM default plug-in.

- Working without The Oracle TSAM default plug-in

If the Oracle TSAM default plug-in is removed from the 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 framework service to get a correlation ID. [Listing 3-12](#) displays an ID generation example.

Listing 3-12 ID Generation Example

```
extern int _TMDLLENTY 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 correlation ID generation routine of TSAM framework, *libtsam* must be linked with the plug-in.

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 only when service execution is completed.

A Basic Implementation

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

```

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

```

Check Commonly Used Metrics

Table 3-8 Service Monitoring Plug-in Routine Metrics

Field Name	Type	Description
TA_MONMSGWAITTIME	long	<p>The request message waiting time in server's request IPC queue before execution.</p> <p>The unit is millisecond. The waiting time is computed in two scenarios,</p> <ul style="list-style-type: none"> • Call Path Monitoring is enabled for this message. <p>The waiting time is computed by considering the last time stamp of transport to this service. The waiting time is exact.</p> <ul style="list-style-type: none"> • No Call Path Monitoring is enabled. <p>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.</p>
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 seconds since time seconds since epoch. It is used with TA_MONLASTTIMESEC

Table 3-8 Service Monitoring Plug-in Routine Metrics

TA_MONERRNO	long	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 framework. Plug-in can also get the current time and the last time timestamp.
TA_MONSVCNAME	string	The service name.
TA_MONLOCATION	string	The process location of current process. It has same meaning in call path monitoring.

System Server Monitoring Plug-in Routine

Oracle TSAM supports two Tuxedo built-in servers monitoring, GWTDOMAIN and BRIDGE. 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 framework. The interval is specified by “sysinterval” policy of TMMONITOR specification. Data collection occurs on the on-going server operations.

A Basic Implementation

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

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

Check Commonly Used Metrics

Table 3-9 System Server Monitoring Plug-in Routine Metrics

Field Name	Type	Description
TA_MONLOCATION	string	The process location of current process. It has same meaning in call path monitoring.
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 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.

Transaction Monitoring Plug-in Routine

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

A Basic Implementation

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

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

Check Commonly Used Metrics

[Listing 3-10](#) lists the commonly used transaction monitoring plug-in routine metrics.

Table 3-10 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.

Configure the Plug-in to Tuxedo

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

Register to Tuxedo

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

[Listing 3-13](#) shows how the `epifreg` command is used to invoke a plug-in.

Listing 3-13 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/mongui \  
        -a InterceptionSeq=abc/tuxedo/tsam
```

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

1. Using “`epifreg`” to register the custom implementation to 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 supports a Fan-out plug-in mechanism which means multiple plug-ins can work together. Oracle TSAM 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 Tuxedo default interceptor plug-in implementation ID, “bea/performance/mongui”, 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.

Un-register from 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/mongui
```

Note: It is strongly recommended to register/unregister/modify the plug-in after shutting down a Tuxedo application.

Oracle TSAM Agent Plug-in Development/Deployment Notes

- Do not use use Tuxedo ATMI calls in the plug-in except for the FML32 operations `tpalloc/tprealloc/tpfree` and `tpypes`. The monitoring points are embedded in the Tuxedo communication framework. Embedded ATMI calls may compromise current Tuxedo context.
- You do not need to free FML32 buffers; the Oracle TSAM framework will free them. You can add fields as needed. If there is no memory space in the buffer, `tprealloc` must be used to extend the buffer space.

Note: Changed buffers are passed to the plug-in invocation sequence that is after the current one.

- If there is any information returned to the TSAM framework, such as new correlation ID and changed required fields, the latest plug-in changes take effect.

- When using the Oracle TSAM default plug-in, do not set required fields using TMMONITOR and the change them in the plug-in.
- Do not change the MONITORCTL udata. It is a read only interception of application messages. Any modification will result un-expected behavior.