HP Performance Agent

For the UNIX® Operating System

Software Version: 4.70

Tracking Your Transactions

Document Release Date: September 2007
Software Release Date: September 2007

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

HP-UX Release 11.11 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

UNIX® is a registered trademark of The Open Group.

Linux® is a U.S. registered trademark of Linus Torvalds.

Sun Solaris® is a registered trademark of Sun Microsystems, Inc. in the United States and other countries.

IBM and AIX are registered trademarks of International Business Machines Corporation in the United States and other countries.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Support

You can visit the HP Software Support web site at:

www.hp.com/go/hpsoftwaresupport

HP Software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Contents

1 What is Transaction Tracking?

Improving Performance Management

You can improve your ability to manage system performance with the transaction tracking capability of HP Performance Agent and HP GlancePlus.

HP Performance Manager in this document refers only to version 4.0 and beyond for UNIX and Windows platforms. HP Performance Manager 3.x (PerfView) will connect to Performance Agent 4.5 on all platforms, except for Performance Agent 4.5 on the Linux platform. In the future, connectivity to Performance Manager 3.x will be discontinued.

As the number of distributed mission-critical business applications increases, application and system managers need more information to tell them how their distributed information technology (IT) is performing.

- Has your application stopped responding?
- Is the application response time unacceptable?
- Are your service level objectives (SLOs) being met?

The transaction tracking capabilities of Performance Agent and GlancePlus allow IT managers to build in end-to-end manageability of their client/server IT environment in business transaction terms. With Performance Agent, you can define what a business transaction is and capture transaction data that makes sense in the context of *your* business.

When your applications are instrumented with the standard Application Response Measurement (ARM) API calls, these products provide extensive transaction tracking and end-to-end management capabilities across multi-vendor platforms.

Benefits of Transaction Tracking

- Provides a client view of elapsed time from the beginning to the end of a transaction.
- Provides transaction data.
- Helps you manage service level agreements (SLAs).

These topics are discussed in more detail in the remainder of this section.

Client View of Transaction Times

Transaction tracking provides you with a client view of elapsed time from the beginning to the end of a transaction. When you use transaction tracking in your Information Technology (IT) environment, you see the following benefits:

- You can accurately track the number of times each transaction executes.
- You can see how long it takes for a transaction to complete, rather than approximating the time as happens now.
- You can correlate transaction times with system resource utilization.
- You can use your own business deliverable production data in system management applications, such as data used for capacity planning, performance management, accounting, and charge-back.
- You can accomplish application optimization and detailed performance troubleshooting based on a real unit of work (your transaction), rather than representing actual work with abstract definitions of system and network resources.

Transaction Data

When Application Response Measurement (ARM) API calls have been inserted in an application to mark the beginning and end of each business transaction, you can then use the following resource and performance monitoring tools to monitor transaction data:

- Performance Agent provides the registration functionality needed to log, report, and detect alarms on transaction data. Transaction data can be viewed in Performance Manager, Glance, or by exporting the data from Performance Agent log files into files that can be accessed by spreadsheet and other reporting tools.
- Performance Manager graphs performance data for short-term troubleshooting and for examining trends and long-term analysis.
- Glance displays detailed real time data for monitoring your systems and transactions moment by moment.
- Performance Manager, Glance, or the HP Operations Manager message browser allow you to monitor alarms on service level compliance.

Individual transaction metrics are described in Chapter 5, Transaction Metrics

Service Level Objectives

Service level objectives (SLOs) are derived from the stated service levels required by business application users. SLOs are typically based on the development of the service level agreement (SLA). From SLOs come the actual metrics that Information Technology resource managers need to collect, monitor, store, and report on to determine if they are meeting the agreed upon service levels for the business application user.

An SLO can be as simple as monitoring the response time of a simple transaction or as complex as tracking system availability.

A Scenario: Real Time Order Processing

Imagine a successful television shopping channel that employs hundreds of telephone operators who take orders from viewers for various types of merchandise. Assume that this enterprise uses a computer program to enter the order information, check merchandise availability, and update the stock inventory. We can use this fictitious enterprise to illustrate how transaction tracking can help an organization meet customer commitments and SLOs.

Based upon the critical tasks, the customer satisfaction factor, the productivity factor, and the maximum response time, resource managers can determine the level of service they want to provide to their customers.

Chapter 6, Transaction Tracking Examples contains a pseudocode example of how ARM API calls can be inserted in a sample order processing application so that transaction data can be monitored with Performance Agent and Glance.

Requirements for Real Time Order Processing

To meet SLOs in the real time order processing example described above, resource managers must keep track of the length of time required to complete the following critical tasks:

- Enter order information
- Query merchandise availability
- Update stock inventory

The key customer satisfaction factor for customers is how quickly the operators can take their order.

The key productivity factor for the enterprise is the number of orders that operators can complete each hour.

To meet the customer satisfaction and productivity factors, the response times of the transactions that access the inventory database, adjust the inventory, and write the record back must be monitored for compliance to established SLOs. For example, resource managers may have established an SLO for this application that 90 percent of the transactions must be completed in five seconds or less.

Preparing the Order Processing Application

ARM API calls can be inserted into the order processing application to create transactions for `inventory response` and `update inventory`. Note that the ARM API calls must be inserted by application programmers *prior* to compiling the application. See Chapter 6, Transaction Tracking Examples for an example order processing program (written in pseudocode) that includes ARM API calls that define various transactions.

For more information on instrumenting applications with ARM API calls, see the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`). `InstallDir` is the directory in which Performance Agent is installed. The Performance Agent installation directory (`InstallDir`) is as follows for the following operating systems:

- IBM AIX- `/usr/lpp/perf`
- All other UNIX platforms- `/opt/perf`

Monitoring Transaction Data

When an application that is instrumented with ARM API calls is installed and running on your system, you can monitor transaction data with Performance Agent, GlancePlus, or Performance Manager.

... with Performance Agent

Using Performance Agent, you can collect and log data for named transactions, monitor trends in your SLOs over time, and generate alarms when SLOs are exceeded. Once these trends have been identified, Information Technology costs can be allocated based on transaction volumes. Performance Agent alarms can be configured to activate a technician's pager, so that problems can be investigated and resolved immediately. For more information, see Chapter 7, Advanced Features in the *HP Performance Agent for UNIX User's Manual* (located on UNIX under `/<InstallDir>/paperdocs/ovpa/C/`).

Performance Agent is required for transaction data to be viewed in Performance Manager.

... with Performance Manager

Performance Manager receives alarms and transaction data from Performance Agent. For example, you can configure Performance Agent so that when an order processing application takes too long to check stock, Performance Manager receives an alarm and sends a warning to the resource manager's console as an alert of potential trouble.

In Performance Manager, you can select **TRANSACTION** from the Class List window for a data source, then **graph transaction metrics** for various transactions. For more information, see Performance Manager online help.

...with GlancePlus

Use GlancePlus to monitor up-to-the-second transaction response time and whether or not your transactions are performing within your established SLOs. GlancePlus helps you identify and resolve resource bottlenecks that

may be impacting transaction performance. For more information, see the GlancePlus online help, which is accessible through the GlancePlus Help menu.

Guidelines for Using ARM

Instrumenting applications with the ARM API requires some careful planning. In addition, managing the environment that has ARMed applications in it is easier if the features and limitations of ARM data collection are understood. Here is a list of areas that could cause some confusion if they are not fully understood.

1. In order to capture ARM metrics, `ttd` and `midaemon` must be running. For Performance Agent, the `scopeux` collector must be running to log ARM metrics. The `ovpa start` script starts all required processes. Likewise, Glance starts `ttd` and `midaemon` if they are not already active. (See Transaction Tracking Daemon (ttd) in Chapter 2.)
2. Re-read the transaction configuration file, `ttd.conf`, to capture any newly-defined transaction names. (See Transaction Configuration File (ttd.conf) in Chapter 2.)
3. Performance Agent, user applications, and `ttd` must be restarted to capture any *new* or modified transaction ranges and service level objectives (SLOs). (See Adding New Applications in Chapter 2.)
4. Strings in user-defined metrics are ignored by Performance Agent. Only the first six non-string user-defined metrics are logged. (See How Data Types Are Used in Chapter 7.)
5. Using dashes in the transaction name has limitations if you are specifying an alarm condition for that transaction. (See "... with Performance Agent" in the section Alarms in Chapter 3)
6. Performance Agent will only show the first 60 characters in the application name and transaction name. (See Specifying Application and Transaction Names in Chapter 2.)
7. Limit the number of unique transaction names that are instrumented. (See Limits on Unique Transactions in Chapter 3.)
8. Do not allow ARM API function calls to affect the execution of an application from an end-user perspective. (See ARM API Call Status Returns in Chapter 2.)
9. Use shared libraries for linking. (See the section C Compiler Option Examples by Platform in Appendix A)

2 How Transaction Tracking Works

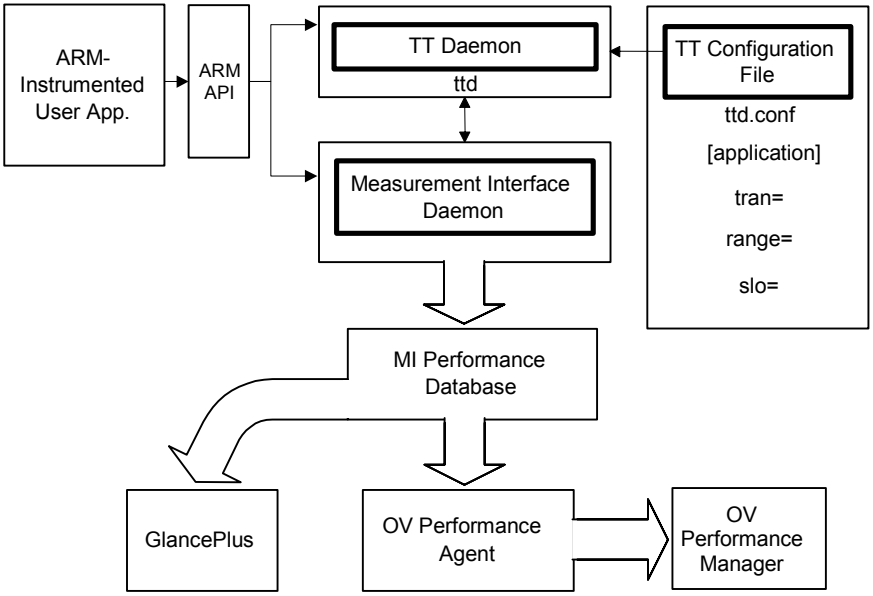
Technical Overview

The following components of Performance Agent and GlancePlus work together to help you define and track transaction data from applications instrumented with Application Response Measurement (ARM) calls.

- The Measurement Interface daemon, `midaemon`, is a daemon process that monitors and reports transaction data to its shared memory segment where the information can be accessed and reported by Performance Agent, Performance Manager, and GlancePlus. On HP-UX systems, the `midaemon` also monitors system performance data.
- The transaction configuration file, `/var/opt/perf/ttd.conf`, is used to define transactions and identify the information to monitor for each transaction.
- The Transaction Tracking daemon, `ttd`, reads, registers, and synchronizes transaction definitions from the transaction configuration file, `ttd.conf`, with the `midaemon`.

These components are shown in Figure 1 and are described in more detail in the remainder of this chapter.

Figure 1 Technical Overview of Transaction Tracking



Support of ARM 2.0

ARM 2.0 is a superset of the previous version of Application Response Measurement. The new features that ARM 2.0 provides are user-defined metrics, transaction correlation, and a logging agent. Performance Agent and GlancePlus support user-defined metrics and transaction correlation but *do not* support the logging agent.

However, you may want to use the logging agent to test the instrumentation in your application. The source code for the logging agent, `logagent.c`, is included in the ARM 2.0 Software Developers Kit (SDK) that is available from the following web site:

http://regions.cmg.org/regions/cmgarmw

For information about using the logging agent, see the *Application Response Measurement 2.0 API Guide*, which is available on UNIX under `/<InstallDir>/paperdocs/arm/C/`. `InstallDir` is the directory in which Performance Agent is installed.

The Application Response Measurement 2.0 API Guide uses the term "application-defined metrics" instead of "user-defined metrics".

Support of ARM API Calls

The Application Response Measurement (ARM) API calls listed below are supported in Performance Agent and GlancePlus.

`arm_init()`	Names and registers the application and (optionally) the user.
`arm_getid()`	Names and registers a transaction class, and provides related transaction information. Defines the context for user-defined metrics.
`arm_start()`	Signals the start of a unique transaction instance.
`arm_update()`	Updates the values of a unique transaction instance.
`arm_stop()`	Signals the end of a unique transaction instance.
`arm_end()`	Signals the end of the application.

See your current *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`) and the *arm (3)* man page for information on instrumenting applications with ARM API calls as well as complete descriptions of the calls and their parameters. For commercial applications, check the product documentation to see if the application has been instrumented with ARM API calls.

For important information about required libraries, see the Appendix A later in this manual.

arm_complete_transaction Call

In addition to the ARM 2.0 API standard, the HP ARM agent supports the `arm_complete_transaction` call. This call, which is an HP-specific extension to the ARM standard, can be used to mark the end of a transaction that has completed when the start of the transaction could not be delimited by an `arm_start` call. The `arm_complete_transaction` call takes as a parameter the response time of the completed transaction instance.

In addition to signaling the end of a transaction instance, additional information about the transaction can be provided in the optional data buffer. See the *arm (3)* man page for more information on this optional data as well a complete description of this call and its parameters.

Sample ARM-Instrumented Applications

For examples of how ARM API calls are implemented, see the sample ARM-instrumented applications, `armsample1.c`, `armsample2.c`, `armsample3.c`, and `armsample4.c`, and their build script, `Make.armsample`, in the `/<InstallDir>/examples/arm/` directory.

- `armsample1.c` shows the use of simple standard ARM API calls.
- `armsample2.c` also shows the use of simple standard ARM API calls. It is similar in structure to `armsample1.c`, but is interactive.
- `armsample3.c` provides examples of how to use the user-defined metrics and the transaction correlator, provided by version 2.0 of the ARM API. This example simulates a client/server application where both server and client perform a number of transactions. (Normally application client and server components would exist in separate programs, but they are put together for simplicity.)

  The client procedure starts a transaction and requests an ARM correlator from its `arm_start` call. This correlator is saved by the client and passed to the server so that the server can use it when it calls `arm_start`. The performance tools running on the server can then use this correlator information to distinguish the different clients making use of the server.

  Also shown in this program is the mechanism for passing user-defined metric values into the ARM API. This allows you to not only see the response times and service-level information in the performance tools, but also data which may be important to the application itself. For example, a transaction may be processing different size requests, and the size of the request could be a user-defined metric. When the response times are high, this user-defined metric could be used to see if long response times correspond to bigger size transaction instances.
- `armsample4.c` provides an example of using user-defined metrics in ARM calls. Different metric values can be passed through `arm_start`, `arm_update`, and `arm_stop` calls. Alternatively, `arm_complete_transaction` can be used where a `tran` cannot be delimited by `start/stop` calls.

Specifying Application and Transaction Names

Although ARM allows a maximum of 128 characters each for application and transaction names in the `arm_init` and `arm_getid` API calls, Performance Agent shows *only* a maximum of 60 characters. All characters beyond the first 60 will not be visible. However, GlancePlus allows you to view up to 128 characters.

Performance Agent applies certain limitations to how application and transaction names are shown in extracted or exported transaction data. These rules also apply to viewing application and transaction names in Performance Manager.

The application name *always* takes precedence over the transaction name. For example, if you are exporting transaction data that has a 65-character application name and a 40-character transaction name, *only* the application name is shown. The last five characters of the application name are not visible.

For another example, if an application name contains 32 characters and the transaction name has 40 characters, Performance Agent shows the entire application name but the transaction name appears truncated. A total of 60 characters are shown. Fifty-nine characters are allocated to the application and transaction names and one character is allocated to the underscore (_) that separates the two names. This is how the application name "`WarehouseInventoryApplication`" and the transaction name "`CallFromWestCoastElectronicSupplier`" would appear in Performance Agent or Performance Manager:

```
WarehouseInventoryApplication_CallFromWestCoastElectronicSup
```

The 60-character combination of application name and transaction name must be unique if the data is to be viewed with Performance Manager.

Transaction Tracking Daemon (ttd)

The Transaction Tracking daemon, `ttd`, reads, registers, and synchronizes transaction definitions from `ttd.conf` with `midaemon`.

`ttd` is started when you start up Performance Agent's `scopeux` data collector with the `ovpa start` command. `ttd` runs in background mode when dispatched, and errors are written to the file `/var/opt/perf/status.ttd`.

`midaemon` must also be running to process the transactions and to collect performance metrics associated with these transactions (see next page).

We strongly recommend that you do not stop `ttd`.

If you must stop `ttd`, any ARM-instrumented applications that are running *must* also be stopped before you restart `ttd` and Performance Agent processes. `ttd` must be running to capture all `arm_init` and `arm_getid` calls being made on the system. If `ttd` is stopped and restarted, transaction IDs returned by these calls will be repeated, thus invalidating the ARM metrics

Use the `ovpa` script to start Performance Agent processes to ensure that the processes are started in the correct order. `ovpa stop` will *not* shut down `ttd`. If `ttd` must be shut down for a reinstall of any performance software, use the command `/<InstallDir>/bin/ttd -k`. However, we do not recommend that you stop `ttd`, except when reinstalling Performance Agent.

If Performance Agent is not on your system, GlancePlus starts `midaemon`. `midaemon` then starts `ttd` if it is not running *before* `midaemon` starts processing any measurement data.

See the `ttd` man page for complete program options.

ARM API Call Status Returns

The `ttd` process must always be running in order to register transactions. If `ttd` is killed for any reason, while it is not running, `arm_init` or `arm_getid` calls will return a “failed” return code. If `ttd` is subsequently restarted, new `arm_getid` calls may re-register the same transaction IDs that are already being used by other programs, thus causing invalid data to be recorded.

When ttd is killed and restarted, ARM-instrumented applications may start getting a return value of -2 (TT_TTDNOTRUNNING) and an EPIPE errno error on ARM API calls. When your application initially starts, a client connection handle is created on any initial ARM API calls. This client handle allows your application to communicate with the ttd process. When ttd is killed, this connection is no longer valid and the next time your application attempts to use an ARM API call, you may get a return value of TT_TTDNOTRUNNING. This error reflects that the *previous* ttd process is no longer running even though there is another ttd process running. (Some of the ARM API call returns are explained in the *arm (3)* man page.)

To get around this error, you must restart your ARM-instrumented applications if ttd is killed. First, stop your ARMed applications. Next, restart ttd (using /<InstallDir>/bin/ovpa start or /<InstallDir>/bin/ttd), and then restart your applications. The restart of your application causes the creation of a new client connection handle between your application and the ttd process.

Some ARM API calls will not return an error if the midaemon has an error. For example, this would occur if the midaemon has run out of room in its shared memory segment. The performance metric GBL_TT_OVERFLOW_COUNT will be > 0. If an overflow condition occurs, you may want to shut down any performance tools that are running (except ttd) and restart the midaemon using the -smdvss option to specify more room in the shared memory segment. (For more information, see the *midaemon* man page.)

We recommend that your applications be written so that they continue to execute even if ARM errors occur. ARM status should not affect program execution.

The number of active client processes that can register transactions with ttd via the arm_getid call is limited to the maxfiles kernel parameter. This parameter controls the number of open files per process. Each client registration request results in ttd opening a socket (an open file) for the RPC connection. The socket is closed when the client application terminates. Therefore, this limit affects only the number of active clients that have registered a transaction via the arm_getid call. Once this limit is reached, ttd will return TT_TTDNOTRUNNING to a client's arm_getid request. The maxfiles kernel parameter can be increased to raise this limit above the number of active applications that will register transactions with ttd.

Measurement Interface Daemon (midaemon)

The Measurement Interface daemon, `midaemon`, is a low-overhead process that continuously collects system performance information. The `midaemon` must be running for Performance Agent to collect transaction data or for GlancePlus to report transaction data. It starts running when you run `scopeux` with the `ovpa start` command or when starting GlancePlus.

Performance Agent and GlancePlus require both the midaemon and `ttd` to be running so that transactions can be registered and tracked. The `ovpa` script starts and stops Performance Agent processing, including the `mideamon`, in the correct order. GlancePlus starts the `mideamon`, if it is not already running. The `midaemon` starts `ttd`, if it is not already running.

See the CPU Overhead section later in this manual for information on the `midaemon` CPU overhead.

See the *midaemon* man page for complete program options.

Transaction Configuration File (ttd.conf)

The transaction configuration file, `/var/opt/perf/ttd.conf`, allows you to define the application name, transaction name, the performance distribution ranges, and the service level objective you want to meet for each transaction. The `ttd` reads `ttd.conf` to determine how to register each transaction.

Customization of `ttd.conf` is optional. The default configuration file that ships with Performance Agent causes *all* transactions instrumented in any application to be monitored.

If you are using a commercial application and don't know which transactions have been instrumented in the application, collect some data using the default `ttd.conf` file. Then look at the data to see which transactions are available. You can then customize the transaction data collection for that application by modifying `ttd.conf`.

Adding New Applications

If you add new ARMed applications to your system that use the default `slo` and `range` values from the `tran=*` line in your `ttd.conf` file, you don't need to do anything to incorporate these new transactions. (See the Configuration File Keywords section for descriptions of `tran`, `range`, and `slo`.) The new transactions will be picked up automatically. The `slo` and `range` values from the `tran` line in your `ttd.conf` file will be applied to the new transactions.

Adding New Transactions

After making additions to the `ttd.conf` file, you must perform the following steps to make the additions effective:

- Stop all applications.
- Execute the `ttd -hup -mi` command as **`root`**.

The above actions cause the `ttd.conf` file to be re-read and registers the new transactions, along with their `slo` and `range` values, with `ttd` and the `midaemon`. The re-read will not change the `slo` or `range` values for any transactions that were in the `ttd.conf` file prior to the re-read.

Changing the Range or SLO Values

If you need to change the SLO or range values of existing transactions in the `ttd.conf` file, you must do the following:

- Stop all ARMed applications.
- Stop the `scopeux` collector using **`ovpa stop`**.
- Stop any usage of Glance.
- Stop the `ttd` by issuing the command **`ttd -k`**.
- Once you have made your changes to the `ttd.conf` file:
- Restart `scopeux` using **`ovpa start`**.
- Restart your ARMed applications.

Configuration File Keywords

The `/var/opt/perf/ttd.conf` configuration file associates transaction names with transaction attributes that are defined by the keywords in Table 1.

Table 1 Configuration File Keywords

Keyword	Syntax	Usage
`tran`	**`tran=`***transaction_name*	Required
`slo`	**`slo=`***sec*	Optional
`range`	**`range=`***sec [,sec,...]*	Optional

These keywords are described in more detail below.

tran

Use tran to define your transaction name. This name must correspond to a transaction that is defined in the `arm_getid` API call in your instrumented application. You must use the tran keyword before you can specify the optional attributes `range` or `slo`. `tran` is the only required keyword within the

configuration file. A trailing asterisk (*) in the transaction name causes a wild card pattern match to be performed when registration requests are made against this entry. Dashes can be used in a transaction name. However, spaces cannot be used in a transaction name.

The transaction name can contain a maximum of 128 characters. However, only the first 60 characters are visible in Performance Agent. GlancePlus can display 128 characters in specific screens.

The default `ttd.conf` file contains several entries. The first entries define transactions used by the Performance Agent data collector `scopeux`, which is instrumented with ARM API calls. The file also contains the entry `tran=*`, which registers all other transactions in applications instrumented with ARM API or Transaction Tracker API calls.

range

Use `range` to specify the transaction performance distribution ranges. Performance distribution ranges allow you to distinguish between transactions that take different lengths of time to complete and to see how many successful transactions of each length occurred. The ranges that you define appear in the GlancePlus Transaction Tracking window.

Each value entered for `sec` represents the upper limit in seconds for the transaction time for the range. The value may be an integer or real number with a maximum of six digits to the right of the decimal point. On HP-UX, this allows for a precision of one microsecond (.000001 seconds). On other platforms, however, the precision is ten milliseconds (0.01 seconds), so only the first two digits to the right of the decimal point are recognized.

A maximum of ten ranges are supported for each transaction you define. You can specify up to nine ranges. One range is reserved for an overflow range, which collects data for transactions that take longer than the largest user-defined range. If you specify more than nine ranges, the first nine ranges are used and the others are ignored.

If you specify fewer than nine ranges, the first unspecified range becomes the overflow range. Any remaining unspecified ranges are not used. The unspecified range metrics are reported as 0.000. The first corresponding unspecified count metric becomes the overflow count. Remaining unspecified count metrics are always zero (0).

Ranges must be defined in ascending order (see examples later in this chapter).

slo

Use `slo` to specify the service level objective (SLO) in seconds that you want to use to monitor your performance service level agreement (SLA).

As with the `range` keyword, the value may be an integer or real number, with a maximum of six digits to the right of the decimal point. On HP-UX, this allows for a precision of one microsecond (.000001 seconds). On other platforms, however, the precision is ten milliseconds (0.01 seconds), so only the first two digits to the right of the decimal point are recognized.

Note that even though transactions can be sorted with one microsecond precision on HP-UX, transaction times are displayed with 100 microsecond precision.

Configuration File Format

The `ttd.conf` file can contain two types of entries: general transactions and application-specific transactions.

General transactions should be defined in the `ttd.conf` file before any application is defined. These transactions will be associated with all the applications that are defined. The default `ttd.conf` file contains one general transaction entry and entries for the `scopeux` collector that is instrumented with ARM API calls.

```
tran=* range=0.5, 1, 2, 3, 5, 10, 30, 120, 300 slo=5.0
```

Optionally, each application can have its own set of transaction names. These transactions will be associated *only* with that application. The application name you specify must correspond to an application name defined in the `arm_init` API call in your instrumented application. Each group of application-specific entries must begin with the name of the application enclosed in brackets. For example:

```
[AccountRec]
tran=acctOne range=0.01, 0.03, 0.05
```

The application name can contain a maximum of 128 characters. However, only the first 60 characters are visible in Performance Agent. Glance can display 128 characters in specific screens.

If there are transactions that have the same name as a “general” transaction, the transaction listed under the application will be used.

For example:

```
tran=abc range=0.01, 0.03, 0.05 slo=0.10
tran=xyz range=0.02, 0.04, 0.06 slo=0.08
tran=t* range=0.01, 0.02, 0.03

[AccountRec}
tran=acctOne range=0.04, 0.06, 0.08
tran=acctTwo range=0.1, 0.2
tran=t* range=0.03, 0.5

[AccountPay]

[GenLedg]
tran=GenLedgOne range=0.01
```

In the example above, the first three transactions apply to all of the three applications specified.

The application `[AccountRec]` has the following transactions: `acctOne`, `acctTwo`, `abc`, `xyz`, and `t*`. One of the entries in the general transaction set also has a wild card transaction named "`t*`". In this case, the "`t*`" transaction name for the `AccountRec` application will be used; the one in the general transaction set is ignored.

The application [`AccountPay`] has only transactions from the general transactions set.

The application [`GenLedg`] has transactions `GenLedgOne`, `abc`, `xyz`, and `t*`.

The ordering of transactions names makes no difference within the application.

For additional information about application and transaction names, see the section Specifying Application and Transaction Names in this chapter.

Configuration File Examples

Example 1

```
tran=* range=0.5,1,2,3,5,10,30,12,30 slo=5.0
```

The "*" entry is used as the default if none of the entries match a registered transaction name. These defaults can be changed on each system by modifying the "*" entry. If the "*" entry is missing, a default set of registration parameters are used that match the initial parameters assigned to the "*" entry above.

Example 2

```
[MANufactr]
tran=MFG01 range=1,2,3,4,5,10 slo=3.0
tran=MFG02 range=1,2.2,3.3,4.0,5.5,10 slo=4.5
tran=MFG03
tran=MFG04 range=1,2.2,3.3,4.0,5.5,10
```

Transactions for the `MANufctr` application, `MFG01`, `MFG02`, and `MFG04`, each use their own unique parameters. The `MFG03` transaction does not need to track time distributions or service level objectives so it does not specify these parameters.

Example 3

```
[Financial]
tran=FIN01
tran=FIN02 range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0
tran=FIN03 range=0.1,0.5,1,2,3,4,5,10,20 slo=2.0
```

Transactions for the `Financial` application, `FIN02` and `FIN03`, each use their own unique parameters. The `FIN01` transaction does not need to track time distributions or service level objectives so it does not specify these parameters.

Example 4

```
[PERSONL]
tran=PERS*  range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0
tran=PERS03 range=0.1,0.2,0.5,1,2,3,4,5,10,20 slo=0.8
```

The `PERS03` transaction for the `PERSONL` application uses its own unique parameters while the remainder of the personnel transactions use a default set of parameters unique to the `PERSONL` application.

Example 5

```
[ACCOUNTS]
tran=ACCT_*    slo=1.0
tran=ACCT_REC range=0.5,1,2,3,4,5,10,20 slo=2.0
tran=ACCT_PAY range=0.5,1,2,3,4,5,10,20 slo=2.0
```

Transactions for the ACCOUNTS application, ACCT_REC and ACCT_PAY, each use their own unique parameters while the remainder of the accounting transactions use a default set of parameters unique to the accounting application. Only the accounts payable and receivable transactions need to track time distributions. The order of transaction names makes no difference within the application.

For more configuration file examples, see Transaction Tracking Examples in Chapter 6

Overhead Considerations for Using ARM

The current versions of Performance Agent and GlancePlus contain modifications to their measurement interface that support additional data required for ARM 2.0. These modifications can result in increased overhead for performance management. You should be aware of overhead considerations when planning ARM instrumentation for your applications.

The overhead areas are discussed in the remainder of this chapter.

Guidelines

Here are some guidelines to follow when instrumenting your applications with the ARM API:

- The total number of separate transaction IDs should be limited to not more than 4,000. Generally, it is cheaper to have multiple instances of the same transaction than it is to have single instances of different transactions. Register *only* those transactions that will be actively monitored.
- Although the overhead for the `arm_start` and `arm_stop` API calls is very small, it can increase when there is a large volume of transaction instances. More than a few thousand `arm_start` and `arm_stop` calls per second on most systems can significantly impact overall performance.
- Request ARM correlators *only* when using ARM 2.0 functionality. (For more information about ARM correlators, see the "Advanced Topics" section in the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`.) The overhead for producing, moving, and monitoring correlator information is significantly higher than for monitoring transactions that are not instrumented to use the ARM 2.0 correlator functionality.
- Larger string sizes (applications registering lengthy transaction names, application names, and user-defined string metrics) will impose additional overhead.

Disk I/O Overhead

The performance management software does not impose a large disk overhead on the system. Glance generally does not log its data to disk. Performance Agent's collector daemon, `scopeux`, generates disk log files, but their size is not significantly impacted by ARM 2.0. The `logtran scopeux` log file is used to store ARM data. For more information, see Chapter 2, "Managing Data Collection" in the *HP Performance Agent for UNIX User's Manual* (located on UNIX under `/<InstallDir>/paperdocs/ovpa/C/`).

CPU Overhead

A program instrumented with ARM calls will generally not run slower because of the ARM calls. This assumes that the rate of `arm_getid` calls is lower than one call per second, and the rate of `arm_start` and `arm_stop` calls is lower than a few thousand per second. More frequent calls to the ARM API should be avoided.

Most of the additional CPU overhead for supporting ARM is incurred inside of the performance tool programs and daemons themselves. The `midaemon` CPU overhead rises slightly but not more than two percent more than it was with ARM 1.0. If the `midaemon` has been requested to track per-transaction resource metrics, the overhead per transaction instance may be twice as high as it would be without tracking per-transaction resource metrics. (You can enable the tracking of per-transaction resource metrics by setting the `log transaction=resource` flag in the `parm` file.) In addition, Glance and `scopeux` CPU overhead will be slightly higher on a system with applications instrumented with ARM 2.0 calls. Only those applications that are instrumented with ARM 2.0 calls that make extensive use of correlators and/or user-defined metrics will have a significant performance impact on the `midaemon`, `scopeux`, or Glance.

An `midaemon` overflow condition can occur when usage exceeds the available default shared memory. This results in:

- No return codes from the ARM calls once the overflow condition occurs.
- Display of incorrect metrics, including blank process names.
- Errors being logged in `status.mi` (for example, "out of space").

Memory Overhead

Programs that are making ARM API calls will not have a significant impact in their memory virtual set size, except for the space used to pass ARM 2.0 correlator and user-defined metric information. These buffers, which are explained in the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir/paperdocs/arm/C/`), should not be a significant portion of a process's memory requirements.

There is additional virtual set size overhead in the performance tools to support ARM 2.0. The `midaemon` process creates a shared memory segment where ARM data is kept internally for use by Performance Agent and GlancePlus. The size of this shared memory segment has grown, relative to the size on releases with ARM 1.0, to accommodate the potential for use by ARM 2.0. By default on most systems, this shared memory segment is approximately 11 megabytes in size. This segment is not all resident in physical memory unless it is required. Therefore, this should not be a significant impact on most systems that are not already memory-constrained. The memory overhead of `midaemon` can be tuned using special startup parameters (see the *midaemon* man page).

3 Getting Started

Putting It All Together

This chapter gives you the information you need to begin tracking transactions and your service level objectives. For detailed reference information, see Chapter 2, How Transaction Tracking Works. See Chapter 6, Transaction Tracking Examples for examples.

Before you start

Performance Agent provides the `libarm.*` shared library in the following locations:

Platform	Path
IBM RS/6000	`/usr/lpp/perf/lib/`
Other UNIX platforms	`/opt/perf/lib/`

If you do not have Performance Agent installed on your system and if `libarm.*` doesn't exist in the path indicated above for your platform, see ARM NOP Library in Appendix A at the end of this manual. See also "The ARM Shared Library (`libarm`)" section in the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`) for information on how to obtain it. `InstallDir` is the directory in which Performance Agent is installed.

For a description of `libarm`, see ARM Library (libarm) in Appendix A at the end of this manual.

Setting Up Transaction Tracking

Follow the procedure below to set up transaction tracking for your application. These steps are described in more detail in the remainder of this section.

1. Define SLOs by determining what key transactions you want to monitor and the response level you expect (*optional*).
2. To monitor transactions in Performance Agent and Performance Manager, make sure that the Performance Agent `parm` file has transaction logging turned on. Then start or restart Performance Agent to read the updated `parm` file.

   Editing the `parm` file is *not* required to see transactions in GlancePlus. However, ttd *must* be running in order to see transactions in GlancePlus. Starting GlancePlus will automatically start `ttd`.
3. Run the application that has been instrumented with ARM API calls that are described in this manual and the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`).
4. Use Performance Agent or Performance Manager to look at the collected transaction data, or use GlancePlus to view current data. If the data isn't visible in Performance Manager, close the data source and then reconnect to it.
5. Customize the configuration file, `ttd.conf`, to modify the way transaction data for the application is collected (*optional*).
6. After making additions to the `ttd.conf` file, you must perform the following steps to make the additions effective:

   a. Stop all ARMed applications.

   b. Execute the `ttd -hup -mi` command as **`root`**.

   These actions re-read the `ttd.conf` file and registers new transactions along with their `slo` and `range` values with `ttd` and the `midaemon`. The re-read will not change the `slo` or `range` values for any transactions that were in the `ttd.conf` file prior to the re-read.
7. If you need to change the `slo` or `range` values of existing transactions in the `ttd.conf` file, do the following:

   a. Stop all ARMed applications.

b Stop the `scopeux` collector using `ovpa stop`.

c Stop all usage of Glance.

d Stop `ttd` using **`ttd -k`**.

Once you have made your changes:

a Restart `scopeux` using **`ovpa start`**.

b Start your ARMed applications.

Defining Service Level Objectives

Your first step in implementing transaction tracking is to determine the key transactions that are required to meet customer expectations and what level of transaction responsiveness is required. The level of responsiveness that is required becomes your service level objective (SLO). You define the service level objective in the configuration file, `ttd.conf`.

Defining service level objectives can be as simple as reviewing your Information Technology department's service level agreement (SLA) to see what transactions you need to monitor to meet your SLA. If you don't have an SLA, you may want to implement one. However, creating an SLA is not required in order to track transactions.

Modifying the Parm File

If necessary, modify the Performance Agent `parm` file to add transactions to the list of items to be logged for use with Performance Manager and Performance Agent. Include the `transaction` option in the `parm` file's log parameter as shown in the following example:

```
log global application process transaction device=disk
```

The default for the `log transaction` parameter is `no resource` and `no correlator`. To turn on resource data collection or correlator data collection, specify `log transaction=resource` or `log transaction=correlator`. Both can be logged by specifying `log transaction=resource, correlator`.

Before you can collect transaction data for use with Performance Agent and Performance Manager, the updated `parm` file must be activated as described below:

Performance Agent status	Command to activate transaction tracking
Running	`ovpa restart`
Not running	`ovpa start`

Collecting Transaction Data

Start up your application. The Transaction Tracking daemon, `ttd`, and the Measurement Interface daemon, `midaemon`, collect and synchronize the transaction data for your application as it runs. The data is stored in the `midaemon`'s shared memory segment where it can be used by Performance Agent or GlancePlus. See Monitoring Performance Data on page 43 for information on using each of these tools to view transaction data for your application.

Error Handling

Due to performance considerations, not all problematic ARM or Transaction Tracker API calls return errors in real time. Some examples of when errors are not returned as expected are:

- calling `arm_start` with a bad `id` parameter such as an uninitialized variable
- calling `arm_stop` without a previously successful `arm_start` call

Performance Agent — To debug these situations when instrumenting applications with ARM calls, run the application long enough to generate and collect a sufficient amount of transaction data. Collect this data with Performance Agent, then use the `extract` program's `export` command to export data from the `logtran` file. Examine the data to see if all transactions were logged as expected. For information about using the `export` command, see Chapters 5 and 6 in the *HP Performance Agent for UNIX User's Manual* (located on UNIX under `/<InstallDir>/paperdocs/ovpa/C/`). Also, check the `/var/opt/perf/status.ttd` file for possible errors.

GlancePlus — To debug these situations when instrumenting applications with ARM calls, run the application long enough to generate a sufficient amount of transaction data, then use GlancePlus to see if all transactions appear as expected.

Limits on Unique Transactions

Depending on your particular system resources and kernel configuration, a limit may exist on the number of unique transactions allowed in your application. This limit is normally several thousand unique `arm_getid` calls.

The number of unique transactions may exceed the limit when the shared memory segment used by `midaemon` is full. If this happens, an overflow message appears in GlancePlus. Although no message appears in Performance Agent, data for subsequent new transactions won't be logged. (However, check `/var/opt/perf/status.scope` for an overflow message.) Data for subsequent new transactions won't be visible in GlancePlus. Transactions that have already been registered will continue to be logged and reported. The `GBL_TT_OVERFLOW_COUNT` metric in GlancePlus reports the number of new transactions that could not be measured.

This situation can be remedied by stopping and restarting the `midaemon` process using the `-smdvss` option to specify a larger shared memory segment size. The current shared memory segment size can be checked using the `midaemon -sizes` command. For more information on optimizing the midaemon for your system, see the *midaemon* man page.

Customizing the Configuration File (optional)

After viewing the transaction data from your application, you may want to customize the transaction configuration file, `/var/opt/perf/ttd.conf`, to modify the way transaction data for the application is collected. This is optional because the default configuration file, `ttd.conf`, will work with all transactions defined in the application. If you do decide to customize the `ttd.conf` file, complete this task on the same systems where you run your application. You must be logged on as **`root`** to modify `ttd.conf`.

See Chapter 2, How Transaction Tracking Works for information on the configuration file keywords – `tran`, `range`, and `slo`. Some examples of how each keyword is used are shown below:

```
tran=Example:     tran=answerid
                  tran=answerid*
                  tran=*

range=Example:    range=2.5,4.2,5.0,10.009

slo=Example:       slo=4.2
```

Customize your configuration file to include all of your transactions and each associated attribute. Note that the use of the `range` or `slo` keyword must be preceded by the `tran` keyword. An example of a `ttd.conf` file is shown below.

```
tran=*
tran=my_first_transaction slo=5.5

[answerid]
tran=answerid1 range=2.5, 4.2, 5.0, 10.009 slo=4.2

[orderid]
tran=orderid1 range=1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0
```

If you need to make additions to the `ttd.conf` file:

- Stop all ARMed applications.
- Execute the `ttd -hup -mi` command as **`root`**.

The above actions re-read the `ttd.conf` file and registers new transactions along with their `slo` and `range` values with `ttd` and the `midaemon`. The re-read will not change the `slo` or `range` value for any transactions that were in the `ttd.conf` file prior to the re-read,

If you need to change the `slo` or `range` values of existing transactions in the `ttd.conf` file, do the following:

1. Stop all ARMed applications.
2. Stop the `scopeux` collector using **`ovpa stop`**.
3. Stop all usage of Glance.
4. Stop `ttd` using **`ttd -k`**.

Once you have made your changes:

1. Restart `scopeux` using **`ovpa start`**.
2. Start your ARMed applications.

Monitoring Performance Data

You can use the following resource and performance management products to monitor transaction data – Performance Agent, Performance Manager, and GlancePlus.

... with Performance Agent

By collecting and logging data for long periods of time, Performance Agent gives you the ability to analyze your system's performance over time and to perform detailed trend analysis. Data from Performance Agent can be viewed with Performance Manager Agent or exported for use with a variety of other performance monitoring, accounting, modeling, and planning tools.

With Performance Agent's `extract` program, data can be exported for use with spreadsheets and analysis programs. Data can also be extracted for archiving and analysis. For more information, see Chapter 5, “Using the Extract Program” and Chapter 6, “Extract Commands” in the *HP Performance Agent for UNIX User's Manual* (located on UNIX under `/<InstallDir>/perf/paperdocs/ovpa/C/`).

Performance Agent and ttd must be running in order to monitor transaction data in Performance Agent. Starting Performance Agent using the ovpa script ensures that the `ttd` and `midaemon` processes that are required to view transaction data in GlancePlus are started in the right order.

... with Performance Manager

Performance Manager imports Performance Agent data and gives you the ability to translate that data into a customized graphical or numerical format. Using Performance Manager, you can perform analysis of historical trends of transaction data and you can perform more accurate forecasting.

You can select **TRANSACTION** from the Class List window for a data source in Performance Manager, then graph transaction metrics for various transactions. For more information, see Performance Manager online help, which is accessible from the Performance Manager Help menu. If you don’t see the transactions you expect in Performance Manager, close the current data source and then reconnect to it.

... with GlancePlus

Monitoring systems with GlancePlus helps you identify resource bottlenecks and provides immediate performance information about the computer system. GlancePlus has a Transaction Tracking window that displays information about all transactions that you have defined and a Transaction Graph window that displays specific information about a single transaction. For example, you can see how each transaction is performing against the SLO that you have defined. For more information about how to use GlancePlus, see the online help that is accessible from the Help menu.

Alarms

You can alarm on transaction data with the following resource and performance management products — Performance Agent, Performance Manager, and GlancePlus.

... with Performance Agent

In order to generate alarms with Performance Agent, you must define alarm conditions in its alarm definitions file, `alarmdef`. You can set up Performance Agent to notify you of an alarm condition in various ways, such as sending an email message or initiating a call to your pager.

To pass a syntax check for the `alarmdef` file, you must have data logged for that application name and transaction name in the log files, or have the names registered in the `ttd.conf` file.

There is a limitation when you define an alarm condition on a transaction that has a dash (–) in its name. To get around this limitation, use the `ALIAS` command in the `alarmdef` file to redefine the transaction name.

For more information about alarms, see Chapter 7, “Performance Alarms”, in the *HP Performance Agent for UNIX User's Manual* (located on UNIX under `/<InstallDir>/paperdocs/ovpa/C/`).

... with Performance Manager

The Performance Manager Monitor option displays alarms generated by Performance Agent. You can view alarm information from all monitored systems and see graphs of metrics that characterize the performance of your systems or applications. Because Performance Manager is designed for both HP-UX and multi-vendor installations, you can receive alarms from a variety of computer systems. When you receive an alarm, you can analyze the details surrounding it by using the color graphs and the tabular backup data provided by Performance Manager. For more information, see Performance Manager online help.

... with GlancePlus

You can configure the Adviser Syntax to alarm on transaction performance. For example, when an alarm condition is met, you can instruct GlancePlus to display information to `stdout`, execute a UNIX command (such as `mailx`), or turn the Alarm button on the main GlancePlus window yellow or red. For more information about alarms in GlancePlus, choose **On This Window** from the Help menu in the Edit Adviser Syntax window.

4 Transaction Tracking Messages

Transaction Tracking Messages

The error codes listed in Table 2 are returned and can be used by the application developer when instrumenting an application with Application Response Measurement (ARM) or Transaction Tracker API calls:

Table 2 Error codes

Error Code	Errno Value	Meaning
-1	EINVAL	Invalid arguments
-2	EPIPE	ttd (registration daemon) not running
-3	ESRCH	Transaction name not found in the ttd.conf file
-4	EOPNOTSUPP	Operating system version not supported

When an application instrumented with ARM or Transaction Tracker API calls is running, return codes from any errors that occur will probably be from the Transaction Tracking daemon, ttd. The Measurement Interface daemon, midaemon, does not produce any error return codes.

If an midaemon error occurs, see the /var/opt/perf/status.mi file for more information.

5 Transaction Metrics

Transaction Metrics

The ARM agent provided as a shared component of both the GlancePlus and Performance Agent, produces many different transaction metrics. To see a complete list of the metrics and their descriptions:

- For installed GlancePlus metrics, use the GlancePlus online help or see the *GlancePlus for HP-UX Dictionary of Performance Metrics* located:
  - On UNIX under `/<InstallDir>/paperdocs/gp/C/` as `metrics.txt` and `metrics.htm`. `InstallDir` is the directory in which Performance Agent is installed.
  - On the web at **http://ovweb.external.hp.com/lpe/doc_serv/**

    Select the "**glanceplus for hpux manuals**" product, select the release version, and click **Search** to see the `metrics.htm` content.
- For installed Performance Agent metrics for specific platforms, see the platform's *HP Performance Agent Dictionary of Operating System Performance Metrics* files located:
  - On UNIX under `/<InstallDir>/paperdocs/ovpa/C/` as `met`*<platform>*`.txt` and `met`*<platform>*`.htm`
  - On the web at **http://ovweb.external.hp.com/lpe/doc_serv/**

    Select the **performance agent for** *<platform>* product, select the release version, and click **Search** to see the `met`*<platform>*`.htm` content.

6 Transaction Tracking Examples

Transaction Tracking Examples

This chapter contains a pseudocode example of how an application might be instrumented with ARM API calls, so that the transactions defined in the application can be monitored with Performance Agent or GlancePlus. This pseudocode example corresponds with the real time order processing scenario described in Chapter 1, What is Transaction Tracking?

Several example transaction configuration files are included in this chapter, including one that corresponds with the real time order processing scenario.

See your current *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`) for information on instrumenting applications with ARM advanced functions such as application-specific metrics (which are called user-defined metrics in Performance Agent and GlancePlus) and transaction correlation. `InstallDir` is the directory in which Performance Agent is installed.

Pseudocode for Real Time Order Processing

This pseudocode example includes the ARM API calls used to define transactions for the real time order processing scenario described in Chapter 1, What is Transaction Tracking? This routine would be processed *each time* an operator answered the phone to handle a customer order. The lines containing the ARM API calls are highlighted with bold text in this example.

```
routine answer calls()
{
*****************************************************
*  Register the transactions if first time in       *
*****************************************************
   if (transactions not registered)
   {
      appl_id = arm_init("Order Processing Application","*", 0,0,0)
      answer_phone_id = arm_getid(appl_id,"answer_phone","1st tran",0,0,0)
      if (answer_phone_id < 0)
         REGISTER OF ANSWER_PHONE FAILED - TAKE APPROPRIATE ACTION
      order_id = arm_getid(appl_id,"order","2nd tran",0,0,0)
      if (order_id < 0)
         REGISTER OF ORDER FAILED - TAKE APPROPRIATE ACTION
      check_id = arm_getid(appl_id,"check_db","3rd tran",0,0,0)
      if (check_id < 0)
         REGISTER OF CHECK DB FAILED - TAKE APPROPRIATE ACTION
      update_id = arm_getid(appl_id,"update","4th tran",0,0,0)
      if (update_id < 0)
         REGISTER OF UPDATE FAILED - TAKE APPROPRIATE ACTION

   } if transactions not registered
*****************************************************
*  Main transaction processing loop
*****************************************************
   while (answering calls)
   {
      if (answer_phone_handle = arm_start(answer_phone_id,0,0,0) < -1)
         TRANSACTION START FOR ANSWER_PHONE NOT REGISTERED
*****************************************************
*  At this point the answer_phone transaction has    *
*  started.  If the customer does not want to order, *
*  end the call; otherwise, proceed with order.     *
*****************************************************
      if (don't want to order)
          arm_stop(answer_phone_handle,ARM_FAILED,0,0,0)
          GOOD-BYE - call complete
      else
```

```
      {
*****************************************************
* They want to place an order - start an order now  *
*****************************************************
         if (order_handle = arm_start(order_id,0,0,0) < -1)
            TRANSACTION START FOR ORDER FAILED
  take order information: name, address, item, etc.
*****************************************************
* Order is complete - end the order transaction     *
*****************************************************
          if (arm_stop(order_handle,ARM_GOOD,0,0,0) < -1)
             TRANSACTION END FOR ORDER FAILED
*****************************************************
*  order taken - query database for availability    *
*****************************************************
         if (query_handle = arm_start(query_id,0,0,0) < -1)
            TRANSACTION QUERY DB FOR ORDER NOT REGISTERED
         query the database for availability
*****************************************************
* database query complete - end query transaction   *
*****************************************************
          if (arm_stop(query_handle,ARM_GOOD,0,0,0) < -1)
             TRANSACTION END FOR QUERY DB FAILED

*****************************************************
* If the item is in stock, process order, and       *
* update inventory.                                 *
*****************************************************
         if (item in stock)
            if (update_handle = arm_start(update_id,0,0,0) < -1)
               TRANSACTION START FOR UPDATE NOT REGISTERED
            update stock
*****************************************************
* update complete - end the update transaction      *
*****************************************************
            if (arm_stop(update_handle,ARM_GOOD,0,0,0) < -1)
               TRANSACTION END FOR ORDER FAILED
*****************************************************
* Order complete - end the call transaction         *
*****************************************************
         if (arm_stop(answer_phone_handle,ARM_GOOD,0,0,0) < -1)
            TRANSACTION END FOR ANSWER_PHONE FAILED
      } placing the order
   GOOD-BYE - call complete
   sleep("waiting for next phone call...zzz...")
   } while answering calls
   arm_end(appl_id, 0,0,0)
} routine answer calls
```

Configuration File Examples

This section contains some examples of the transaction configuration file, `/var/opt/perf/ttd.conf`. For more information on the `ttd.conf` file and the configuration file keywords, see Chapter 2, How Transaction Tracking Works

Example 1 (for Order Processing Pseudocode Example)

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=* range=0.5,1,1.5,2,3,4,5,6,7 slo=1
tran=answer_phone* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=order* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=query_db* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
```

Example 2

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=* range=1,2,3,4,5,6,7,8 slo=5

# The entry below is for the only transaction being
# tracked in this application.  The "*" has been inserted
# at the end of the tran name to catch any possible numbered
# transactions.  For example "First_Transaction1",
# "First_Transaction2", etc.

tran=First_Transaction* range=1,2.2,3.3,4.0,5.5,10 slo=5.5
```

Example 3

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=*
tran=Transaction_One range=1,10,20,30,40,50,60 slo=30
```

Example 4

```
tran=FactoryStor* range=0.05, 0.10, 0.15 slo=3

# The entries below shows the use of an application name.
# Transactions are grouped under the application name.  This
# example also shows the use of less than 10 ranges and
# optional use of "slo."

[Inventory]
tran=In_Stock range=0.001, 0.004, 0.008
tran=Out_Stock range=0.001, 0.005
tran=Returns range=0.1, 0.3, 0.7

[Pers]
tran=Acctg range=0.5, 0.10, slo=5
tran=Time_Cards range=0.010, 0.020
```

7 Advanced Features

Overview

This chapter describes how Performance Agent uses the following ARM 2.0 API features:

- data types
- user-defined metrics
- `scopeux` instrumentation

How Data Types Are Used

Table 3 describes how data types are used in Performance Agent. It is a supplement to "Data Type Definitions" in the "Advanced Topics" section of the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`). `InstallDir` is the directory in which Performance Agent is installed.

Table 3 Data type usage in Performance Agent

`ARM_Counter32`	Data is logged as a 32-bit integer.
`ARM_Counter64`	Data is logged as a 32-bit integer with type casting.
`ARM_CntrDivr32`	Makes the calculation and logs the result as a 32-bit integer.
`ARM_Gauge32`	Data is logged as a 32-bit integer.
`ARM_Gauge64`	Data is logged as a 32-bit integer with type casting.
`ARM_GaugeDivr32`	Makes the calculation and logs the result as a 32-bit integer.
`ARM_NumericID32`	Data is logged as a 32-bit integer.
`ARM_NumericID64`	Data is logged as a 32 bit integer with type casting.
`ARM_String8`	Ignored.
`ARM_String32`	Ignored.

Performance Agent does not log string data. Because Performance Agent logs data every five minutes, and what is logged is the summary of the activity for that interval, it cannot summarize the strings provided by the application.

Performance Agent logs the Minimum, Maximum, and Average for the first six usable user-defined metrics. If your ARM-instrumented application passes a `Counter32`, a `String8`, a `NumericID 32`, a `Gauge32`, a `Gauge64`, a `Counter64`, a `NumericID64`, a `String32`, and a `GaugeDivr32`, Performance Agent logs the `Min`, `Max`, and `Average` over the five-minute interval for the

`Counter32`, `NumericID32`, `Gauge32`, `Gauge64`, `NumericID32` and `NumericID64` as 32-bit integers. The `String8` and `String32` are ignored because strings cannot be summarized in Performance Agent. The `GaugeDivr32` is also ignored because only the first six usable user-defined metrics are logged. (For more examples, see the next section, User-Defined Metrics.)

User-Defined Metrics

This section is a supplement to “Application-Defined Metrics” under “Advanced Topics” in the *Application Response Measurement 2.0 API Guide* (located on UNIX under `/<InstallDir>/paperdocs/arm/C/`). It contains some examples about how Performance Agent handles user-defined metrics (referred to as application-defined metrics in ARM). The examples in Table 4 show what is logged if your program passes the following data types.

Table 4 Examples of What is Logged with Specific Program Data Types

...what your program passes in	...what is logged
EXAMPLE 1	
`String8`	
`Counter32`	`Counter32`
`Gauge32`	`Gauge32`
`CntrDivr32`	`CntrDivr32`
EXAMPLE 2	
`String32`	
`NumericID32`	`NumericID32`
`NumericID64`	`NumericID64`

Table 4 Examples of What is Logged with Specific Program Data Types (cont'd)

...what your program passes in	...what is logged
EXAMPLE 3 NumericID32 String8 NumericID64 Gauge32 String32 Gauge64	 NumericID32 NumericID64 Gauge32 Gauge64
EXAMPLE 4 String8 String32	(nothing)
EXAMPLE 5 Counter32 Counter64 CntrDivr32 Gauge32 Gauge64 NumericID32 NumericID64	 Counter32 Counter64 CntrDivr32 Gauge32 Gauge64 NumericID32

Because Performance Agent cannot summarize strings, no strings are logged.

In example 1, only the counter, gauge, and counter divisor are logged.

In example 2, only the numerics are logged.

In example 3, only the numerics and gauges are logged.

In example 4, nothing is logged.

In example 5, because only the first six user-defined metrics are logged, NumericID64 is not logged.

Scopeux Instrumentation

The `scopeux` data collector has been instrumented with ARM API calls. When Performance Agent starts, `scopeux` automatically starts logging two transactions, `Scope_Get_Process_Metrics` and `Scope_Get_Global_Metrics`. Both transactions will be in the HP Performance Tools application.

Transaction data is logged every five minutes so you will find that five `Get Process` transactions (one transaction per minute) have completed during each interval. The `Scope_Get_Process_Metrics` transaction is instrumented around the processing of process data. If there are 200 processes on your system, the `Scope_Get_Process_Metrics` transaction should take longer than if there are only 30 processes on your system.

The `Scope_Get_Global_Metrics` transaction is instrumented around the gathering of all five-minute data, including global data. This includes `global`, `application`, `disk`, `transaction`, and other data types.

To stop the logging of process and global transactions data, remove or comment out the entries for the `scopeux` transactions in the `ttd.conf` file.

A Appendix

Overview

This appendix discusses:

- the Application Response Measurement library (`libarm`)
- C compiler option examples by platform
- the Application Response Measurement NOP library (`libarmNOP`)
- using Java wrappers

ARM Library (libarm)

With Performance Agent and GlancePlus, the environment is set up to make it easy to compile and use the ARM facility. The libraries needed for development are located in `/opt/perf/lib/`. See the next section in this appendix for specific information about compiling.

The library files listed in Table 5 exist on an HP-UX 11.11 and beyond Performance Agent and GlancePlus installation:

Table 5 HP-UX 11.11 and Beyond Performance Agent and GlancePlus Library Files

`/opt/perf/lib/`	`libarm.0`	HP-UX 10.X compatible shared library for ARM (not thread safe). If you execute a program on HP-UX 11 that was linked on 10.20 with `-larm`, the 11.0 loader will automatically reference this library.
	`libarm.1`	HP-UX 11 compatible shared library (thread safe). This will be referenced by programs that were linked with `-larm` on HP-UX releases. If a program linked on 10.20 references this library, (for example, if it was not linked with `-L /opt/perf/lib`, it may abort with an error such as "`/usr/lib/dld.sl: Unresolved symbol: _thread_once (code) from libtt.sl`".
	`libarm.sl`	A symbolic link to `libarm.1`
	`libarmNOP.sl`	"No-operation" shared library for ARM (the API calls succeed but do nothing; used for testing and on systems that do not have Performance Agent installed.

Table 5 HP-UX 11.11 and Beyond Performance Agent and GlancePlus Library Files (cont'd)

/opt/perf/lib/pa20_64/	Note that these files will be referenced automatically by programs compiled on HP-UX 11 with the +DD64 compiler option.	
	libarm.sl	64-bit shared library for ARM.
	libarmNOP.sl	64-bit "no-operation" shared library for ARM (the API calls succeed but do nothing; used for testing and on systems that do not have Performance Agent installed.
/usr/lib/pa20_64/	libarm.sl	Symbolic link to /opt/perf/lib/pa20_64/libarm.sl

The additional library files listed in Table 6 exist on an IA64 HP-UX installation:

Table 6 HP-UX IA64 Library Files

/opt/perf/lib/hpux32/	libarm.so.1	IA64/32-bit shared library for ARM.
/opt/perf/lib/hpux64/	libarm.so.1	IA64/64-bit shared library for ARM.

Because the ARM library makes calls to HP-UX that may change from one version of the operating system to the next, programs should link with the shared library version, using -larm. Compiling an application that has been instrumented with ARM API calls and linking with the archived version of the ARM library (-Wl, -a archive) is not supported. (For additional information, see Transaction Tracking Daemon (ttd) on page 23 in Chapter 2.

The library files that exist on an AIX operating system with Performance Agent and GlancePlus installation are as follows.

Table 7 AIX Library Files

/usr/lpp/perf/lib/	libarm.a	32-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.a	32-bit shared library for ARM. This library is used for testing on systems that do not have Performance Agent installed.

The library files that exist on a Solaris operating system with Performance Agent and GlancePlus installation are as follows.

Table 8 Solaris Library Files for 32-bit programs

/opt/perf/lib/	libarm.so	32-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	32-bit shared library for ARM. This library is used for testing on systems that do not have Performance Agent installed.

Table 9 Solaris Library Files for 64-bit programs

/opt/perf/lib/sparc_64/	libarm.so	64-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	libarmNOP.so	64-bit shared library for ARM. This library is used for testing on systems that do not have Performance Agent installed.

You must compile 64-bit programs using `-xarch=generic64` command-line parameter along with the other parameters provided for 32-bit programs.

The library files that exist on a Linux operating system with Performance Agent and GlancePlus installation are as follows.

Table 10 Linux Library Files

`/opt/perf/lib/`	`libarm.so`	32-bit shared ARM library (thread safe). This library is referenced by programs linked with -larm.
	`libarmNOP.so`	32-bit shared library for ARM. This library is used for testing on systems that do not have Performance Agent installed.

C Compiler Option Examples by Platform

The `arm.h` include file is located in `/opt/perf/include/`. For convenience, this file is accessible via a symbolic link from `/usr/include/` as well. This means that you do not need to use "`-I/opt/perf/include/`" (although you may). Likewise, `libarm` resides in `/opt/perf/lib/` but is linked from `/usr/lib/`. You should always use "`-L/opt/perf/lib/`" when building ARMed applications.

- For Linux:

  The following example shows a compile command for a C program using the ARM API.

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L
  -Xlinker -rpath -Xlinker /opt/perf/lib
  ```

- For HP-UX:

  For HP-UX releases 11.2x on IA64 platforms, change the `-L` parameter from `-L/opt/perf/lib to -L/opt/perf/lib/hpux32` for 32-bit IA ARMed program compiles, and to `-L/opt/perf/lib/hpux64` for 64-bit IA program compiles using ARM.

  The following example shows a compile command for a C program using the ARM API.

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib
  -larm
  ```

- For Sun Solaris:

  The following example works for Performance Agent and GlancePlus on Sun Solaris:

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib
  -larm -lnsl
  ```

- For 64-bit programs on Sun Solaris:

  The following example works for Performance Agent and 64-bit programs on Sun Solaris:

  ```
  cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include
  -L /opt/perf/lib/sparc_64 -larm -lnsl
  ```

- For IBM AIX:

The file placement on IBM AIX differs from the other platforms (`/usr/lpp/perf/` is used instead of `/opt/perf/`), therefore the example for IBM AIX is different from the examples of other platforms:

```
cc myfile.c -o myfile -I /usr/lpp/perf/include -L /usr/lpp/perf/lib -larm
```

For C++ compilers, the `-D_PROTOTYPES` flag may need to be added to the compile command in order to reference the proper declarations in the `arm.h` file.

ARM NOP Library

The “no-operation” library (named `libarmNOP.*` where `*` is `sl`, `so`, or `a`, depending on the OS platform) is shipped with Performance Agent and Glance. This shared library does nothing except return valid status for every ARM API call. This allows an application that has been instrumented with ARM to run on a system where Performance Agent or GlancePlus is not installed.

To run your ARM-instrumented application on a system where Performance Agent or GlancePlus is not installed, copy the NOP library and name it `libarm.sl` (`libarm.so`, or `libarm.a` depending on the platform) in the appropriate directory (typically, `/<InstallDir>/lib/`). When Performance Agent or GlancePlus is installed, it will overwrite this NOP library with the correct functional library (which is not removed as the other files are). This ensures that instrumented programs will not abort when Performance Agent or GlancePlus is removed from the system.

Using the Java Wrappers

The Java Native Interface (JNI) wrappers are functions created for your convenience to allow the Java applications to call the HP ARM2.0 API. These wrappers (`armapi.jar`) are included with the ARM sample programs located in the `/<InstallDir>/examples/arm/` directory. `InstallDir` is the directory in which Performance Agent is installed.

Examples

Examples of the Java wrappers are located in the `/<InstallDir>/examples/arm/` directory. This location also contains a README file, which explains the function of each wrapper.

Setting Up an Application (arm_init)

To set up a new application, make a new instance of ARMApplication and pass the name and the description for this API. Each application needs to be identified by a unique name. The ARMApplication class uses the C – function `arm_init`.

Syntax:

```
ARMApplication myApplication =
new ARMApplication("name","description");
```

Setting Up a Transaction (arm_getid)

To set up a new transaction, you can choose whether or not you want to use user-defined metrics (UDMs). The Java wrappers use the C – function `arm_getid`.

Setting Up a Transaction With UDMs

If you want to use UDMs, you must first define a new ARMTranDescription. ARMTranDescription builds the Data Buffer for `arm_getid`. (See also the `jprimeudm.java` example.)

Syntax:

```
ARMTranDescription myDescription =
new ARMTranDescription("transactionName","details");
```

If you don't want to use details, you can use another constructor:

Syntax:

```
ARMTranDescription myDescription =
new ARMTranDescription("transactionName");
```

Adding the Metrics

Metric 1-6:

Syntax:

```
myDescription.addMetric(metricPosition, metricType,
metricDescription);
```

Parameters:

`metricPosition: 1-6`

```
metricType: ARMConstants.ARM_Counter32
ARMConstants.ARM_Counter64 ARMConstants.ARM_CntrDivr32
ARMConstants.ARM_Gauge32 ARMConstants.ARM_Gauge64
ARMConstants.ARM_GaugeDivr32 ARMConstants.ARM_NumericID32
ARMConstants.ARM_NumericID64 ARMConstants.ARM_String8
```

Metric 7:

Syntax:

```
myDescription.addStringMetric("description");
```

Then you can create the Transaction:

Syntax:

```
myApplication.createTransaction(myDescription);
```

Setting the Metric Data

Metric 1-6:

Syntax:

```
myTransaction.setMetricData(metricPosition, metric);
```

Examples for "Metric"

```
ARMGauge32Metric metric = new ARMGauge32Metric(start);
ARMCounter32Metric metric = new ARMCounter32Metric(start);
ARMCntrDivr32Metric metric = new ARMCntrDivr32Metric(start,
1000);
```

Metric 7:

Syntax:

```
myTransaction.setStringMetricData(text);
```

Setting Up a Transaction Without UDMs

When you set up a transaction without UDMs, you can immediately create the new transaction. You can choose whether or not to specify details.

With Details

Syntax:

```
ARMTransaction myTransaction =
myApplication.createTransaction("Transactionname","detail
s";
```

Without Details

Syntax:

```
ARMTransaction myTransaction =
myApplication.createTransaction("Transactionname");
```

Setting Up a Transaction Instance

To set up a new transaction instance, make a new instance of ARMTransactionInstance with the method `createTransactionInstance()` of ARMTransaction.

Syntax:

```
ARMTransactionInstance myTranInstance =
myTransaction.createTransactionInstance();
```

Starting a Transaction Instance (arm_start)

To start a transaction instance, you can choose whether or not to use correlators. The following methods call the C – function `arm_start` with the relevant parameters.

Starting the Transaction Instance Using Correlators

When you use correlators, you must distinguish between getting and delivering a correlator.

Requesting a Correlator

If your transaction instance wants to request a correlator, the call is as follows (see also the `jcorrelators.java` example).

Syntax:

```
int status = myTranInstance.startTranWithCorrelator();
```

Passing the Parent Correlator

If you already have a correlator from a previous transaction and you want to deliver it to your transaction, the syntax is as follows:

Syntax

```
int status = startTran(parent);
```

Parameter

`parent` is the delivered correlator. In the previous transaction, you can get the transaction instance correlator with the method `getCorrelator()`.

Requesting and Passing the Parent Correlator

If you already have a correlator from a previous transaction and you want to deliver it to your transaction and request a correlator, the syntax is as follows:

Syntax:

```
int status =
myTranInstance.startTranWithCorrelator(parent);
```

Parameter:

`parent` is the delivered correlator. In the previous transaction, you can get the transaction instance correlator with the method `getCorrelator()`.

Retrieving the Correlator Information

You can retrieve the transaction instance correlator using the `getCorrelator()` method as follows:

Syntax:

```
ARMTranCorrelator parent =
myTranInstance.getCorrelator();
```

Starting the Transaction Instance Without Using Correlators

When you do not use correlators, you can start your transaction instance as follows:

Syntax:

```
int status = myTranInstance.startTran();
```

`startTran` returns a unique handle to status, which is used for the update and stop.

Updating Transaction Instance Data

You can update the UDMs of your transaction instance any number of times between the start and stop. This part of the wrappers calls the C – function `arm_update` with the relevant parameters.

Updating Transaction Instance Data With UDMs

When you update the data of your transaction instance with UDMs, first, you must set the new data for the metric. For example,

`metric.setData(value)  for ARM_Counter32 ARM_Counter64, ARM_Gauge32, ARM_Gauge64, ARM_NumericID32, ARM_NumericID64`

`metric.setData(value,value) for ARM_CntrDivr32 and , ARM_GaugeDivr32`

`metric.setData(string)  for ARM_String8 and ARM_String32`

Then you can set the metric data to new (like the examples in the Setting the Metric Data section) and call the update:

Syntax:

```
myTranInstance.updateTranInstance();
```

Updating Transaction Instance Data Without UDMs

When you update the data of your transaction instance without UDMs, you just call the update. This sends a “heartbeat” indicating that the transaction instance is still running.

Syntax:

```
myTranInstance.updateTranInstance();
```

Providing a Larger Opaque Application Private Buffer

If you want to use the second buffer format, you must pass the byte array to the update method. (See the *Application Response Measurement 2.0 API Guide*, p. 31, located on UNIX under `/<InstallDir>/paperdocs/arm/C/`.). `InstallDir` is the directory in which OVPA is installed.

Syntax:

```
myTranInstance.updateTranInstance(byteArray);
```

Stopping the Transaction Instance (arm_stop)

To stop the transaction instance, you can choose whether or not to stop it with or without a metric update.

Stopping the Transaction Instance With a Metric Update

To stop the transaction instance with a metric update, call the method `stopTranInstanceWithMetricUpdate`.

Syntax:

```
myTranInstance.stopTranInstanceWithMetricUpdate
(transactionCompletionCode);
```

Parameter:

The transaction Completion Code can be:

`ARMConstants. ARM_GOOD.`	Use this value when the operation ran normally and as expected.
`ARMConstants.ARM_ABORT.`	Use this value when there is a fundamental failure in the system.
`ARMConstants.ARM_FAILED.`	Use this value in applications where the transaction worked properly, but no result was generated.

These methods use the C – function `arm_stop` with the requested parameters.

Stopping the Transaction Instance Without a Metric Update

To stop the transaction instance without a metric update, you can use the method `stopTranInstance`.

Syntax:

```
myTranInstance.stopTranInstance(transactionCompletionCode
);
```

Using Complete Transaction

The Java wrappers can use the `arm_complete_transaction` call. This call can be used to mark the end of a transaction that has lasted for a specified number of nanoseconds. This enables the real time integration of transaction response times measured outside of the ARM agent.

In addition to signaling the end of a transaction instance, additional information about the transaction (UDMs) can be provided in the optional data buffer.

(See also the jcomplete.java example.)

Using Complete Transaction With UDMs:

Syntax:

```
myTranInstance.completeTranWithUserData(status,responseTi
me;
```

Parameters:

`status`	• `ARMConstants. ARM_GOOD` Use this value when the operation ran normally and as expected. • `ARMConstants.ARM_ABORT` Use this value when there was a fundamental failure in the system. • `ARMConstants.ARM_FAILED` Use this value in applications where the transaction worked properly, but no result was generated.
`responseTime`	This is the response time of the transaction in nanoseconds.

Using Complete Transaction Without UDMs:

Syntax:

```
myTranInstance.completeTran(status,responseTime);
```

Further Documentation

For further information about the Java classes, see the doc folder in the `/<InstallDir>/examples/arm/` directory, which includes html-documentation for every Java class. Start with `index.htm`.

Glossary

alarm

A signal that an alarm event has occurred. The signal can be either a notification or an automatically triggered action. The event can be a predefined condition that has been met or was exceeded.

client

A system that requests a service from a server. In the context of diskless clusters, a client uses the server's disks and has none of its own. In the context of NFS, a client mounts file systems that physically reside on another system (the Network File System server).

export

An `extract` program command that copies log file data from the performance application to an external file format for use by other programs (such as spreadsheets and word processors).

extract

A Performance Agent program that helps you manage your data. In `extract` mode, raw or previously extracted log files can be read in, reorganized or filtered as desired, and the results are combined into a single, easy-to-manage extracted log file. In `export` mode, raw or previously extracted log files can be read in, reorganized or filtered as desired, and the results are written to class-specific exported data files for use in spreadsheets and analysis programs.

interval

Specific time periods during which performance data is gathered.

log files

Performance measurement data files that are either raw or extracted. Raw log files contain summarized measurements of system data. Extracted log files contain a user-defined subset of data extracted from a raw log file.

measurement interface

A set of proprietary library calls used by the performance applications to obtain performance data.

metric

A specific system measurement that helps you characterize performance.

midaemon

The Management Interface process that monitors system performance and creates counters from system event traces that are read and displayed by performance applications.

MI shared memory segment

The interface between the kernel and the performance collectors. The midaemon translates trace data into this shared memory segment where it can be accessed by Performance Agent, GlancePlus, and Performance Manager. (Also known as the MI Performance Database.)

performance distribution range

An amount of time that you define with the `range= keyword` in the transaction configuration file, `ttd.conf`.

resource manager

A company's Information Technology (IT) manager who monitors service levels between IT and other business sections of a company.

scopeux

The Performance Agent program that collects performance data and writes (logs) the raw measurement data to log files for later analysis or archiving.

service level agreement (SLA)

A document prepared for each mission- and business-critical application that explicitly defines the service levels IT is expected to deliver to users. It

specifies what the user group can expect from the IT community in terms of system response, quantities of work, and system availability.

service level objective (SLO)

Objectives that identify what the IT staff must do to support the terms of the SLA, how it will monitor the provisions, and what it will do when an exception occurs. SLAs are not required for a company to implement SLOs.

transaction

Some amount of work performed by a computer system on behalf of a user. The boundaries of this work are defined by the user.

ttd

A transaction daemon that reads, registers, and synchronizes transaction definitions from the `ttd.conf` file with the midaemon.

Index

E

G

I

J

K

L

M

N

O

P

R

S

T

U

V